Network Working Group
Request for Comments: 926

ISO
December 1984

Protocol for Providing the Connectionless-Mode Network Services

(Informally - ISO IP)

ISO DIS 8473

Status of this Memo:

This document is distributred as an RFC for information only. It does not specify a standard for the ARPA-Internet. Distribution of this memo is unlimited.

Note:

This document has been prepared by retyping the text of ISO DIS 8473 of May 1984, which is currently undergoing voting within ISO as a Draft International Standard (DIS). Although this RFC has been reviewed after typing, and is believed to be substantially correct, it is possible that typographic errors not present in the ISO document have been overlooked.

Alex McKenzie
BBN

TABLE OF CONTENTS

1 SCOPE AND FIELD OF APPLICATION..... 2

2 REFERENCES..... 3

3 DEFINITIONS..... 4
3.1 Reference Model Definitions..... 4
3.2 Service Conventions Definitions..... 4
3.3 Network Layer Architecture Definitions..... 4
3.4 Network Layer Addressing Definitions..... 5
3.5 Additional Definitions..... 5

4 SYMBOLS AND ABBREVIATIONS..... 7
4.1 Data Units..... 7
4.2 Protocol Data Units..... 7
4.3 Protocol Data Unit Fields..... 7
4.4 Parameters..... 8
4.5 Miscellaneous..... 8

5 OVERVIEW OF THE PROTOCOL..... 9
5.1 Internal Organization of the Network Layer..... 9
5.2 Subsets of the Protocol..... 9
5.3 Addressing..... 10
5.4 Service Provided by the Network Layer..... 10
5.5 Service Assumed from the Subnetwork Service Provider..... 11
5.5.1 Subnetwork Addresses..... 12
5.5.2 Subnetwork Quality of Service..... 12
5.5.3 Subnetwork User Data..... 13
5.5.4 Subnetwork Dependent Convergence Functions..... 13
5.6 Service Assumed from Local Evironment..... 14

6 PROTOCOL FUNCTIONS..... 16
6.1 PDU Composition Function..... 16
6.2 PDU Decomposition Function..... 17
6.3 Header Format Analysis Function..... 17
6.4 PDU Lifetime Control Function..... 18
6.5 Route PDU Function..... 18
6.6 Forward PDU Function..... 19
6.7 Segmentation Function..... 19
6.8 Reassembly Function..... 20
6.9 Discard PDU Function..... 21

6.10 Error Reporting Function..... 22
6.10.1 Overview..... 22
6.10.2 Requirements..... 23
6.10.3 Processing of Error Reports..... 24
6.11 PDU Header Error Detection..... 25
6.12 Padding Function..... 26
6.13 Security..... 26
6.14 Source Routing Function..... 27
6.15 Record Route Function..... 28
6.16 Quality of Service Maintenance Function..... 29
6.17 Classification of Functions..... 29

7 STRUCTURE AND ENCODING OF PDUS..... 32
7.1 Structure..... 32
7.2 Fixed Part..... 34
7.2.1 General..... 34
7.2.2 Network Layer Protocol Identifier..... 34
7.2.3 Length Indicator..... 35
7.2.4 Version/Protocol Identifier Extension..... 35
7.2.5 PDU Lifetime..... 35
7.2.6 Flags..... 36
7.2.6.1 Segmentation Permitted and More Segments Flags. 36
7.2.6.2 Error Report Flag..... 37
7.2.7 Type Code..... 37
7.2.8 PDU Segment Length..... 37
7.2.9 PDUChecksum..... 38
7.3 Address Part..... 38
7.3.1 General..... 38
7.3.1.1 Destination and Source Address Information... 39

7.4 Segmentation Part..... 40
7.4.1 Data Unit Identifier..... 41
7.4.2 Segment Offset..... 41
7.4.3 PDU Total Length..... 41
7.5 Options Part..... 41
7.5.1 General..... 41
7.5.2 Padding..... 43
7.5.3 Security..... 43
7.5.4 Source Routing..... 44
7.5.5 Recording of Route..... 45
7.5.6 Quality of Service Maintenance..... 46
7.6 Priority..... 47

7.7 Data Part..... 47
7.8 Data (DT) PDU..... 49
7.8.1 Structure..... 49
7.8.1.1 Fixed Part..... 50
7.8.1.2 Addresses..... 50
7.8.1.3 Segmentation..... 50
7.8.1.4 Options..... 50
7.8.1.5 Data..... 50
7.9 Inactive Network Layer Protocol..... 51
7.9.1 Network Layer Protocol Id..... 51
7.9.2 Data Field..... 51
7.10 Error Report PDU (ER)..... 52
7.10.1 Structure..... 52
7.10.1.1 Fixed Part..... 53
7.10.1.2 Addresses..... 53
7.10.1.3 Segmentation..... 53
7.10.1.4 Options..... 54
7.10.1.5 Reason for Discard..... 54
7.10.1.6 Error Report Data Field..... 55

8 FORMAL DESCRIPTION..... 56
8.1 Values of the State Variable..... 57
8.2 Atomic Events..... 57
8.2.1 N.UNITDATA_request and N.UNITDATA_indication..... 57
8.2.2 SN.UNITDATA_request and SN.UNITDATA_indication... 58
8.2.3 TIMER Atomic Events..... 59
8.3 Operation of the Finite State Automation..... 59
8.3.1 Type and Constant Definitions..... 61
8.3.2 Interface Definitions..... 65
8.3.3 Formal Machine Definition..... 67

9 CONFORMANCE..... 84
9.1 Provision of Functions for Conformance..... 84

INTRODUCTION

This Protocol is one of a set of International Standards produced to facilitate the interconnection of open systems. The set of standards covers the services and protocols required to achieve such interconnection.

This Protocol Standard is positioned with respect to other related standards by the layers defined in the Reference Model for Open Systems Interconnection (ISO 7498). In particular, it is a protocol of the Network Layer. The Protocol herein described is a Subnetwork Independent Convergence Protocol combined with relay and routing functions as described in the Internal Organization of the Network Layer (ISO iiii). This Protocol provides the connectionless-mode Network Service as defined in ISO 8348/DAD1, Addendum to the Network Service Definition Covering Connectionless-mode Transmission, between Network Service users and/or Network Layer relay systems.

The interrelationship of these standards is illustrated in Figure 0-1 below:

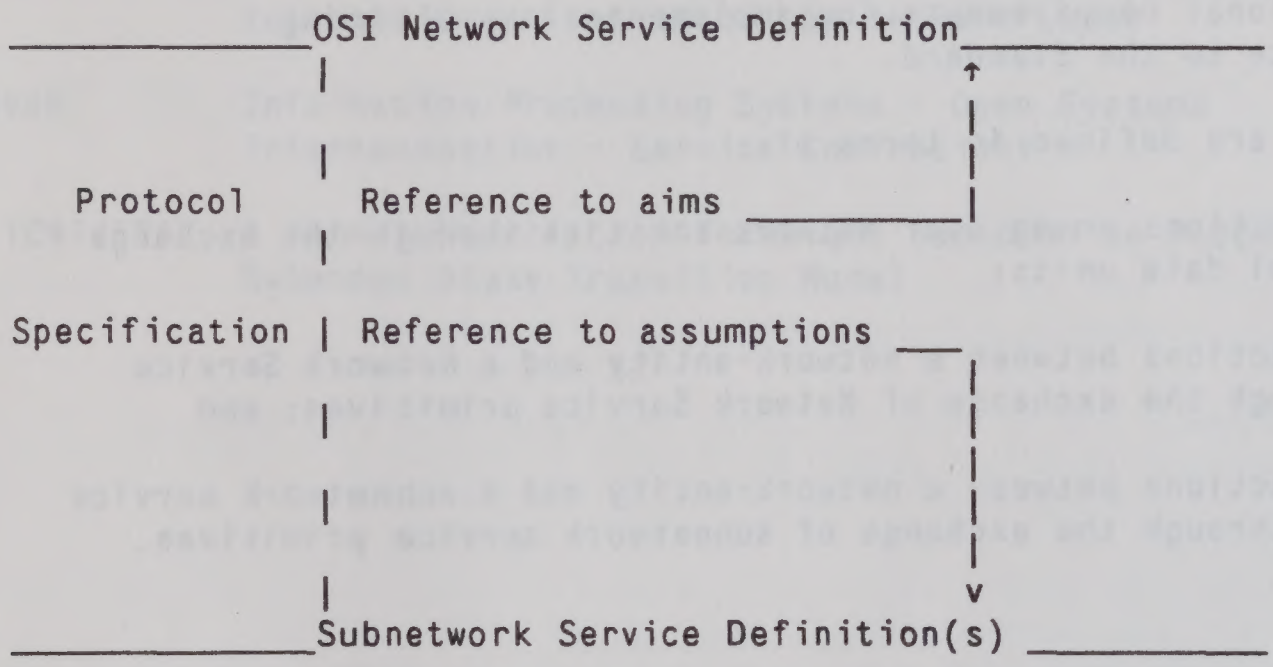


Figure 0-1. Interrelationship of Standards

1 SCOPE AND FIELD OF APPLICATION

This International Standard specifies a protocol which is used to provide the Connectionless-mode Network Service as described in ISO 8348/DAD1, Addendum to the Network Service Definition Covering Connectionless-mode Transmission. The protocol herein described relies upon the provision of a connectionless-mode subnetwork service.

This Standard specifies:

a) procedures for the connectionless transmission of data and control information from one network-entity to a peer network-entity;

b) the encoding of the protocol data units used for the transmission of data and control information, comprising a variable-length protocol header format;

c) procedures for the correct interpretation of protocol control information; and

d) the functional requirements for implementations claiming conformance to the Standard.

The procedures are defined in terms of:

a) the interactions among peer network-entities through the exchange of protocol data units;

b) the interactions between a network-entity and a Network Service user through the exchange of Network Service primitives; and

c) the interactions between a network-entity and a subnetwork service provider through the exchange of subnetwork service primitives.

2 REFERENCES

ISO 7498	Information Processing Systems - Open Systems Interconnection - Basic Reference Model
DP 8524	Information Processing Systems - Open Systems Interconnection - Addendum to ISO 7498 Covering Connectionless-Mode Transmission
DIS 8348	Information Processing Systems - Data Communications - Network Service Definition
ISO 8348/DAD1	Information Processing Systems - Data Communications - Addendum to the Network Service Definition Covering Connectionless-Mode Transmission
ISO 8348/DAD2	Information Processing Systems - Data Communications - Addendum to the Network Service Definition Covering Network Layer Addressing
DP iiii	Information Processing Systems - Data Communications - Internal Organization of the Network Layer
DP 8509	Information Processing Systems - Open Systems Interconnection - Service Conventions
ISO TC97/SC16	A Formal Description Technique based on an N1825 Extended State Transition Model

SECTION ONE. GENERAL

3 DEFINITIONS

3.1 Reference Model Definitions

This document makes use of the following concepts defined in ISO 7498:

a) Network layer
b) Network service
c) Network service access point
d) network service access point address
e) Network entity
f) Routing
f) Service
h) Network protocol
i) Network relay
j) Network protocol data unit
k) End system

3.2 Service Conventions Definitions

This document makes use of the following concepts from the OSI Service Conventions (ISO 8509):

l) Service user
m) Service provider

3.3 Network Layer Architecture Definitions

This document makes use of the following concepts from the Internal Organization of the Network Layer (ISO iiii):

n) Subnetwork

o) Relay system
p) Intermediate system
q) Subnetwork service

3.4 Network Layer Addressing Definitions

This document makes use of the following concepts from DIS 8348/DAD2, Addendum to the Network Service Definition Covering Network layer addressing:

r) Network entity title
s) Network protocol address information
t) Subnetwork address
u) Domain

3.5 Additional Definitions

For the purposes of this document, the following definitions apply:

a) automaton - a machine designed to follow automatically a predetermined sequence of operations or to respond to encoded instructions.

b) local matter - a decision made by a system concerning its behavior in the Network Layer that is not subject to the requirements of this Protocol.

c) segment - part of the user data provided in the N_UNITDATA request and delivered in the N_UNITDATA indication.

d) initial PDU - a protocol data unit carrying the whole of the user data from an N_UNITDATA request.

e) derived PDU - a protocol data unit whose fields are identical to those of an initial PDU, except that it carries only a segment of the user data from an N_UNITDATA request.

f) segmentation - the act of generating two or more derived PDUS from an initial or derived PDU. The derived PDUs together carry the entire user data of the initial or derived PDU from which they were generated. [Note: it is possible that such an initial PDU will never actually be generated for a particular N_UNITDATA request, owing to the immediate application of segmentation.]

g) reassembly - the act of regenerating an initial PDU (in order to issue an N_UNITDATA indication) from two or more derived PDUs produced by segmentation.

4 SYMBOLS AND ABBREVIATIONS

4.1 Data Units

PDU	Protocol Data Unit
NSDU	Network Service Data Unit
SNSDU	Subnetwork Service Data Unit

4.2 Protocol Data Units

DT PDU	Data Protocol Data Unit
ER PDU	Error Report Protocol Data Unit

4.3 Protocol Data Unit Fields

NPID	Network Layer Protocol Identifier
LI	Length Indicator
V/P	Version/protocol Identifier Extension
LT	Lifetime
SP	Segmentation Permitted Flag
MS	More Segments Flag
E/R	Error Report Flag
TP	Type
SL	Segment Length
CS	Checksum
DAL	Destination Address Length
DA	Destination Address
SAL	Source Address Length
SA	Source Address
DUID	Data Unit Identifier
SO	Segment Offset
TL	Total Length

4.4 Parameters

DA	Destination Address
SA	Source Address
QOS	Quality of Service

4.5 Miscellaneous

SNICP	Subnetwork Independent Convergence Protocol
SNDCP	Subnetwork Dependent Convergence Protocol
SNAcP	Subnetwork Access Protocol
SN	Subnetwork
P	Protocol
NSAP	Network Service Access Point
SNSAP	Subnetwork Service Access Point
NPAI	Network Protocol Address Information
NS	Network Service

5 OVERVIEW OF THE PROTOCOL

5.1 Internal Organization of the Network Layer

The architecture of the Network Layer is described in a separate document, Internal Organization of the Network Layer (ISO iiii), in which an OSI Network Layer structure is defined, and a structure to classify protocols as an aid to the progression toward that structure is presented. This protocol is designed to be used in the context of the internetworking protocol approach defined in that document, between Network Service users and/or Network Layer relay systems. As described in the Internal Organization of the Network Layer, the protocol herein described is a Subnetwork Independent Convergence Protocol combined with relay and routing functions designed to allow the incorporation of existing network standards within the OSI framework.

A Subnetwork Independent Convergence Protocol is one which can be defined on a subnetwork independent basis and which is necessary to support the uniform appearance of the OSI Connectionless-mode Network Service between Network Service users and/or Network Layer relay systems over a set of interconnected homogeneous or heterogeneous subnetworks. This protocol is defined in just such a subnetwork independent way so as to minimize variability where subnetwork dependent and/or subnetwork access protocols do not provide the OSI Network Service.

The subnetwork service required from the lower sublayers by the protocol described herein is identified in Section 5.5.

5.2 Subsets of the Protocol

Two proper subsets of the full protocol are also defined which permit the use of known subnetwork characteristics, and are therefore not subnetwork independent.

One protocol subset is for use where it is known that the source and destination end-systems are connected by a single subnetwork. This is known as the "Inactive Network Layer Protocol" subset. A second subset permits simplification of the header where it is known that the source and destination end-systems are connected by subnetworks whose subnetwork service data unit (SNSDU) sizes are greater than or equal to a known bound large enough for segmentation not to be required. This subset, selected by setting the "segmentation permitted" flag to zero, is known as the "non-segmenting" protocol subset.

5.3 Addressing

The Source Address and Destination Address parameters referred to in Section 7.3 of this International Standard are OSI Network Service Access Point Addresses. The syntax and semantics of an OSI Network Service Access Point Address, the syntax and encoding of the Network Protocol Address Information employed by this Protocol, and the relationship between the NSAP and the NPAI is described in a separate document, ISO 8348/DAD2, Addendum to the Network Service Definition covering Network Layer Addressing.

The syntax and semantics of the titles and addresses used for relaying and routing are also described in ISO 8348/DAD2.

5.4 Service Provided by the Network Layer

The service provided by the protocol herein described is a connectionless-mode Network Service. The connectionless-mode Network Service is described in document ISO 8348/DAD1, Addendum to the Network Service Definition Covering Connectionless-mode Transmission. The Network Service primitives provided are summarized below:

Primitives	Parameters
N_UNITDATA Request Indication	NS_Destination_Address, NS_Source_Address, NS_Quality_of_Service, NS_Userdata

Table 5-1. Network Service Primitives

The Addendum to the Network Service Definition Covering Connectionless-mode Transmission (ISO 8348/DAD1) states that the maximum size of a connectionless-mode Network-service-data-unit is limited to 64512 octets.

5.5 Service Assumed from the Subnetwork Service provider

The subnetwork service required to support this protocol is defined as comprising the following primitives:

Primitives	Parameters
SN_UNITDATA Request Indication	SN_Destination_Address, SN_Source_Address, SN_Quality_of_Service, SN_Userdata

Table 5-2. Subnetwork Service Primitives

5.5.1 Subnetwork Addresses

The source and destination addresses specify the points of attachment to a public or private subnetwork(s) involved in the transmission. Subnetwork addresses are defined in the Service Definition of each individual subnetwork.

The syntax and semantics of subnetwork addresses are not defined in this Protocol Standard.

5.5.2 Subnetwork Quality of Service

Subnetwork Quality of Service describes aspects of a subnetwork connectionless-mode service which are attributable solely to the subnetwork service provider.

Associated with each subnetwork connectionless-mode transmission, certain measures of quality of service are requested when the primitive action is initiated. These requested measures (or parameter values and options) are based on a priori knowledge by the Network Service provider of the service(s) made available to it by the subnetwork. Knowledge of the nature and type of service available is typically obtained prior to an invocation of the subnetwork connectionless-mode service.

Note:

The quality of service parameters identified for the subnetwork connectionless-mode service may in some circumstances be directly derivable from or mappable onto those identified in the connectionless-mode Network Service; e.g., the parameters

a) transit delay;
b) protection against unauthorized access;
c) cost determinants;
d) priority; and
e) residual error probability

as defined in ISO 8348/DAD1, Addendum to the Network Service Definition Covering Connectionless-mode Transmission, may be employed.

For those subnetworks which do not inherently provide Quality of Service as a parameter when the primitive action is initiated, it is a local matter as to how the semantics of the service requested might be preserved. In particular, there may be instances in which the Quality of Service requested cannot be maintained. In such circumstances, the subnetwork service provider shall attempt to deliver the protocol data unit at whatever Quality of Service is available.

5.5.3 Subnetwork User Data

The SN_Userdata is an ordered multiple of octets, and is transferred transparently between the specified subnetwork service access points.

The subnetwork service is required to support a subnetwork service data unit size of at least the maximum size of the Data PDU header plus one octet of NS-Userdata. This requires a minimum subnetwork service data unit size of 256 octets.

Where the subnetwork service can support a subnetwork service data unit (SNSDU) size greater than the size of the Data PDU header plus one octet of NS_Userdata, the protocol may take advantage of this. In particular, if all SNSDU sizes of the subnetworks involved are known to be large enough that segmentation is not required, then the "non-segmenting" protocol subset may be used.

5.5.4 Subnetwork Dependent Convergence Functions

Subnetwork Dependent Convergence Functions may be performed to provide a connectionless-mode subnetwork service in the case where subnetworks also provide a connection-oriented subnetwork service. If a subnetwork provides a connection-oriented service, some subnetwork dependent function is assumed to provide a mapping into the required subnetwork service described in the preceding text.

A Subnetwork Dependent Convergence Protocol may also be employed in those cases where functions assumed from the subnetwork service provider are not performed.

5.6 Service Assumed from Local Evironment

A timer service is provided to allow the protocol entity to schedule events.

There are three primitives associated with the S_TIMER service:

1) the S-TIMER request;

2) the S_TIMER response; and

3) the S_TIMER cancel.

The S_TIMER request primitive indicates to the local environment that it should initiate a timer of the specified name and subscript and maintain it for the duration specified by the time parameter.

The S_TIMER response primitive is initiated by the local environment to indicate that the delay requested by the corresponding S_TIMER request primitive has elapsed.

The S_TIMER cancel primitive is an indication to the local environment that the specified timer(s) should be cancelled. If the subscript parameter is not specified, then all timers with the specified name are cancelled; otherwise, the timer of the given name and subscript is cancelled. If no timers correspond to the parameters specified, the local environment takes no action.

The parameters of the S_TIMER service primitives are:

Primitives	Parameters
S_TIMER Request	S_Time S_Name S_Subscript
S_TIMER Response Cancel	S_Name S_Subscript

Table 5-3. Timer Primitives

The time parameter indicates the time duration of the specified timer. An identifying label is associated with a timer by means of the name parameter. The subscript parameter specifies a value to distinguish timers with the same name. The name and subscript taken together constitute a unique reference to the timer.

SECTION TWO. SPECIFICATION OF THE PROTOCOL

6 PROTOCOL FUNCTIONS

This section describes the functions performed as part of the Protocol.

Not all of the functions must be performed by every implementation. Section 6.17 specifies which functions may be omitted and the correct behavior where requested functions are not implemented.

6.1 PDU Composition Function

This function is responsible for the construction of a protocol data unit according to the rules of protocol given in Section 7. Protocol Control Information required for delivering the data unit to its destination is determined from current state information and from the parameters provided with the N_UNITDATA Request; e.g., source and destination addresses, QOS, etc. User data passed from the Network Service user in the N_UNITDATA Request forms the Data field of the protocol data unit.

During the composition of the protocol data unit, a Data Unit Identifier is assigned to identify uniquely all segments of the corresponding NS_Userdata. The "Reassemble PDU" function considers PDUs to correspond to the same Initial PDU, and hence N_UNITDATA request, if they have the same Source and Destination Addresses and Data Unit Identifier.

The Data Unit Identifier is available for ancillary functions such as error reporting. The originator of the PDU must choose the Data Unit Identifier so that it remains unique (for this Source and Destination Address pair) for the maximum lifetime of the PDU (or any Derived PDUs) in the network.

During the composition of the PDU, a value of the total length of the PDU is determined by the originator and placed in the Total Length field of the PDU header. This field is not changed in any Derived PDU for the lifetime of the protocol data unit.

Where the non-segmenting subset is employed, neither the Total Length field nor the Data Unit Identifier field is present. During the composition of the protocol data unit, a value of the total length of the PDU is determined by the originator and placed in the Segment Length field of the PDU header. This field is not changed for the lifetime of the PDU.

6.2 PDU Decomposition Function

This function is responsible for removing the Protocol Control Information from the protocol data unit. During this process, information pertinent to the generation of the N_UNITDATA Indication is retained. The data field of the PDU received is reserved until all segments of the original service data unit have been received; this is the NS_Userdata parameter of the N_UNITDATA Indication.

6.3 Header Format Analysis Function

This function determines whether the full Protocol described in this Standard is employed, or one of the defined proper subsets thereof. If the protocol data unit has a Network Layer Protocol Identifier indicating that this is a standard version of the Protocol, this function determines whether a PDU received has reached its destination using the destination address provided in the PDU is the same as the one which addresses an NSAP served by this network-entity, then the PDU has reached its destination; if not, it must be forwarded.

If the protocol data unit has a Network Layer Protocol Identifier indicating that the Inactive Network Layer Protocol subset is in use, then no further analysis of the PDU header is required. The

network-entity in this case determines that either the network address encoded in the network protocol address information of a supporting subnetwork protocol corresponds to a network Service Access Point address served by this network-entity, or that an error has occurred. If the subnetwork PDU has been delivered correctly, then the protocol data unit may be decomposed according to the procedure described for that particular subnetwork protocol.

6.4 PDU Lifetime Control Function

This function is used to enforce the maximum PDU lifetime. It is closely associated with the "Header Format Analysis" function. This function determines whether a PDU received may be forwarded or whether its assigned lifetime has expired, in which case it must be discarded.

The operation of the Lifetime Control function depends upon the Lifetime field in the PDU header. This field contains, at any time, the remaining lifetime of the PDU (represented in units of 500 Milliseconds). The Lifetime of the Initial PDU is determined by the originating network-entity, and placed in the Lifetime field of the PDU.

6.5 Route PDU Function

This function determines the network-entity to which a protocol data unit should be forwarded, using the destination NSAP address parameters, Quality of Service parameter, and/or other parameters. It determines the subnetwork which must be transited to reach that network-entity. Where segmentation occurs, it further determines which subnetwork(s) the segments may transit to reach that network-entity.

6.6 Forward PDU Function

This function issues a subnetwork service primitive (see Section 5.5) supplying the subnetwork identified by the "Route PDU" function with the protocol data unit as an SNSDU, and the address information required by that subnetwork to identify the "next" intermediate-system within the subnetwork-specific address domain.

When an Error Report PDU is to be forwarded, and is longer than the maximum user data acceptable by the subnetwork, it shall be truncated to the maximum acceptable length ad forwarded with no other change. When a Data PDU is to be forwarded ad is longer than the maximum user data acceptable by the subnetwork, the Segmentation function is applied (See Section 6.7, which follows).

6.7 Segmentation Function

Segmentation is performed when the size of the protocol data unit is greater than the maximum size of the user data parameter field of the subnetwork service primitive.

Segmentation consists of composing two or more new PDUs (Derived PDUs) from the PDU received. The PDU received may be the Initial PDU, or it may be a Derived PDU. The Protocol Control Information required to identify, route, and forward a PDU is duplicated in each PDU derived from the Initial PDU. The user data encapsulated within the PDU received is divided such that the Derived PDUs satisfy the size requirements of the user data parameter field of the subnetwork service primitive.

Derived PDUs are identified as being from the same Initial PDU by means of

a) the source address,

b) the destination address, and

c) the data unit identifier.

The following fields of the PDU header are used in conjunction with the Segmentation function:

a) Segment Offset - identifies at which octet in the data field of the Initial PDU the segment begins;

b) Segment Length - specifies the number of octets in the Derived PDU, including both header and data;

c) More Segments Flag - set to one if this Derived PDU does not contain, as its final octet of user data, the final octet of the Initial PDU; and

d) Total Length - specifies the entire length of the Initial PDU, including both header and data.

Derived PDUs may be further segmented without constraining the routing of the individual Derived PDUs.

A Segmentation Permitted flag is set to one to indicate that segmentation is permitted. If the Initial PDU is not to be segmented at any point during its lifetime in the network, the flag is set to zero.

When the "Segmentation Permitted" flag is set to zero, the non-segmenting protocol subset is in use.

6.8 Reassembly Function

The Reassembly Function reconstructs the Initial PDU transmitted to the destination network-entity from the Derived PDUs generated during the lifetime of the Initial PDU.

A bound on the time during which segments (Derived PDUs) of an Initial PDU will be held at a reassembly point is provided so that resources may be released when it is no longer expected that any outstanding segments of the Initial PDU will arrive at the reassembly point. When such an event occurs, segments (Derived PDUs) of the Initial PDU held at the reassembly point are discarded, the resources allocated for those segments are freed,

and if selected, an Error Report is generated.

Note:

The design of the Segmentation and Reassembly functions is intended principally to be used such that reassembly takes place at the destination. However, other schemes which

a) interact with the routing algorithm to favor paths on which fewer segments are generated,

b) generate more segments than absolutely required in order to avoid additional segmentation at some subsequent point, or

c) allow partial/full reassembly at some point along the route where it is known that the subnetwork with the smallest PDU size has been transited

are not precluded. The information necessary to enable the use of one of these alternative strategies may be made available through the operation of a Network Layer Management function.

While the exact relationship between reassembly lifetime and PDU lifetime is a local matter, the reassembly algorithm must preserve the intent of the PDU lifetime. Consequently, the reassembly function must discard PDUs whose lifetime would otherwise have expired had they not been under the control of the reassembly function.

6.9 Discard PDU Function

This function performs all of the actions necessary to free the resources reserved by the network-entity in any of the following situations (Note: the list is not exhaustive):

a) A violation of protocol procedure has occurred.

b) A PDU is received whose checksum is inconsistent with its contents.

c) A PDU is received, but due to congestion, it cannot be processed.

d) A PDU is received whose header cannot be analyzed.

e) A PDU is received which cannot be segmented and cannot be forwarded because its length exceeds the maximum subnetwork service data unit size.

f) A PDU is received whose destination address is unreachable or unknown.

g) Incorrect or invalid source routing was specified. This may include a syntax error in the source routing field, and unknown or unreachable address in the source routing field, or a path which is not acceptable for other reasons.

h) A PDU is received whose PDU lifetime has expired or the lifetime expires during reassembly.

i) A PDU is received which contains an unsupported option.

6.10 Error Reporting Function

6.10.1 Overview

This function causes the return of an Error Report PDU to the source network-entity when a protocol data unit is discarded. An "error report flag" in the original PDU is set by the source network-entity to indicate whether or not Error Report PDUs are to be returned.

The Error Report PDU identifies the discarded PDU, specifies the type of error detected, and identifies the location at which the error was detected. Part or all of the discarded PDU is included in the data field of the Error Report PDU.

The address of the originator of the Data Protocol Data Unit is

conveyed as both the destination address of the Error Report PDU as well as the source address of the original Data PDU; the latter is contained in the Data field of the Error Report PDU. The address of the originator of the Error Report PDU is contained in the source address field of the header of the Error Report PDU.

Note:

Non-receipt of an Error Report PDU does not imply correct delivery of a PDU issued by a source network-entity.

6.10.2 Requirements

An Error Report PDU shall not be generated to report the discarding of a PDU that itself contains an Error Report.

An Error Report PDU shall not be generated upon discarding of a PDU unless that PDU has the Error Report flag set to allow Error Reports.

If a Data PDU is discarded, and has the Error Report flag set to allow Error Reports, an Error Report PDU shall be generated if the reason for discard (See Section 6.9) is

a) destination address unreachable,

b) source routing failure,

c) unsupported options, or

d) protocol violation.

Note:

It is intended that this list shall include all nontransient reasons for discard; the list may therefore need to be amended or extended in the light of any changes made in the definitions of such reasons.

If a Data PDU with the Error Report flag set to allow Error Reports is discarded for any other reason, an Error Report PDU may be generated (as an implementation option).

6.10.3 Processing of Error Reports

Error Report PDUs are forwarded by intermediate network-entities in the same way as Data PDUs. It is possible that an Error Report PDU may be longer than the maximum user data size of a subnetwork that must be traversed to reach the origin of the discarded PDU. In this case, the Forward PDU function shall truncate the PDU to the maximum size acceptable.

The entire header of the discarded data unit shall be included in the data field of the Error Report PDU. Some or all of the data field of the discarded data unit may also be included.

Note:

Since the suppression of Error Report PDUs is controlled by the originating network-entity and not by the NS User, care should be exercised by the originator with regard to suppressing ER PDUs so that error reporting is not suppressed for every PDU generated.

6.11 PDU Header Error Detection

The PDU Header Error Detection function protects against failure of intermediate or end-system network-entities due to the processing of erroneous information in the PDU header. The function is realized by a checksum computed on the PDU header. The checksum is verified at each point at which the PDU header is processed. If PDU header fields are modified (for example, due to lifetime function), then the checksum is modified so that the checksum remains valid.

An intermediate system network-entity must not recompute the checksum for the entire header, even if fields are modified.

Note:

This is to ensure that inadvertent modification of a header while a PDU is being processed by an intermediate system (for example, due to a memory fault) may still be detected by the PDU Header Error function.

The use of this function is optional, and is selected by the originating network-entity. If the function is not used, the checksum field of the PDU header is set to zero.

If the function is selected by the originating network-entity, the value of the checksum field causes the following formulae to be satisfied:

$$\sum_{i=1}^{L} a_i = 0 \pmod{255}$$

$$\sum_{i=1}^{L} (L-i+1)\, a_i = 0 \pmod{255}$$

Where L = the number of octets in the PDU header, and
a_i = value of octet at position i.

When the function is in use, neither octet of the checksum field may be set to zero.

Annex C contains descriptions of algorithms which may be used to calculate the correct value of the checksum field when the PDU is created, and to update the checksum field when the header is modified.

6.12 Padding Function

The padding function is provided to allow space to be reserved in the PDU header which is not used to support any other function. Octet alignment must be maintained.

Note:

An example of the use of this function is to cause the data field of a PDU to begin on a convenient boundary for the originating network-entity, such as a computer word boundary.

6.13 Security

An issue related to the quality of the network service is the protection of information flowing between transport-entities. A system may wish to control the distribution of secure data by assigning levels of security to PDUs. As a local consideration, the Network Service user could be authenticated to ascertain whether the user has permission to engage in communication at a particular security level before sending the PDU. While no protocol exchange is required in the authentication process, the optional security parameter in the options part of the PDU header may be employed to convey the particular security level between peer network-entities.

The syntax and semantics of the security parameter are not specified by this Standard. The security parameter is related to the "protection from unauthorized access" Quality of service parameter described in ISO 8348/DAD1, Addendum to the Network Service Definition Covering Connectionless-mode Transmission. However, to facilitate interoperation between end-systems and relay-systems by avoiding different interpretations of the same encoding, a mechanism is provided to distinguish user-defined security encoding from standardized security encoding.

6.14 Source Routing Function

The Source Routing function allows the originator to specify the path a generated PDU must take. Source routing can only be selected by the originator of a PDU. Source Routing is accomplished using a list of intermediate system addresses (or titles, see Section 5.3 and 5.5.1) held in a parameter within the options part of the PDU Header. The size of the option field is determined by the originating network-entity. The length of this option does not change as the PDU traverses the network. Associated with this list is an indicator which identifies the next entry in the list to be used; this indicator is advanced by the receiver of the PDU when the next address matches its own address. The indicator is updated as the PDU is forwarded so as to identify the appropriate entry at each stage of relaying.

Two forms of the source routing option are provided. The first form, referred to as complete source routing, requires that the specified path must be taken; if the specified path cannot be taken, the PDU must be discarded. The source may be informed of the discard using the Error Reporting function described in Section 6.10.

The second form is referred to as partial source routing. Again, each address in the list must be visited in the order specified while on route to the destination. However, with this form of source routing the PDU may take any path necessary to arrive at the next address in the list. The PDU will not be discarded (for source routing related causes) unless one of the addresses specified cannot be reached by any available route.

6.15 Record Route Function

The Record Route function permits the exact recording of the paths taken by a PDU as it traverses a series of interconnected subnetworks. A recorded route is composed of a list of intermediate system addresses held in a parameter within the options part of the PDU header. The size of the option field is determined by the originating network-entity. The length of this option does not change as the PDU traverses the network.

The list is constructed as the PDU traverses a set of interconnected subnetworks. Only intermediate system addresses are included in the recorded route. The address of the originator of the PDU is not recorded in the list. When an intermediate system network-entity processes a PDU containing the record route parameter, the system inserts its own address (or titles, see Sections 5.3 or 5.5.1) into the list of recorded addresses.

The record route option contains an indicator which identifies the next available octet to be used for recording of route. This identifier is updated as entries are added to the list. If the addition of the current address to the list would exceed the size of the option field, the indicator is set to show that recording of route has terminated. The PDU may still be forwarded to its final destination, without further addition of intermediate system addresses.

Note:

The Record Route function is principally intended to be used in the diagnosis of network problems. Its mechanism has been designed on this basis, and may provide a return path.

6.16 Quality of Service Maintenance Function

In order to support the Quality of Service requested by Network Service users, the Protocol may need to make QOS information available at intermediate systems. This information may be used by network entities in intermediate systems to make routing decisions where such decisions affect the overall QOS provided to NS users.

In those instances where the QOS indicated cannot be maintained, the NS provider will attempt to deliver the PDU at a QOS less than that indicated. The NS provider will not necessarily provide a notification of failure to meet the indicated quality of service.

6.17 Classification of Functions

Implementations do not have to support all of the functions described in Section 6. Functions are divided into three categories:

Type 1: These functions must be supported.

Type 2: These functions may or may not be supported. If an implementation does not support a Type 2 function, and the function is selected by a PDU, then the PDU shall be discarded, and an Error Report PDU shall be generated and forwarded to the originating network-entity, providing that the Error Report flag is set.

Type 3: These functions may or may not be supported. If an implementation does not support a Type 3 function, and the function is selected by a PDU, then the function is not performed and the PDU is processed exactly as though the function was not selected. The protocol data unit shall not be discarded.

Table 6-1 shows how the functions are divided into these three categories:

Function	Type
PDU Composition	1
PDU Decomposition	1
Header Format Analysis	1
PDU Lifetime Control	1
Route PDU	1
Forward PDU	1
Segment PDU	1
Reassemble PDU	1
Discard PDU	1
Error Reporting	1 (note 1)
PDU Header Error Detection	1 (note 1)
Padding	1 (notes 1 2)
Security	2
Complete Source Routing	2
Partial Source Routing	3
Priority	3
Record Route	3
Quality of Service Maintenance	3

Table 6-1. Categorization of Protocol Functions

Notes:

1) While the Padding, Error Reporting, and Header Error Detection functions must be provided, they are provided only when selected by the sending Network Service user.

2) The correct treatment of the Padding function involves no processing. Therefore, this could equally be described as a Type 3 function.

3) The rationale for the inclusion of type 3 functions is that in the case of some functions it is more important to forward the PDUs between intermediate systems or deliver them to an end-system than it is to support the functions. Type 3 functions should be used in those cases where they are of an advisory nature and should not be the cause of the discarding of a PDU when not supported.

7 STRUCTURE AND ENCODING OF PDUS

7.1 Structure

All Protocol Data Units shall contain an integral number of octets. The octets in a PDU are numbered starting from one (1) and increasing in the order in which they are put into an SNSDU. The bits in an octet are numbered from one (1) to eight (8), where bit one (1) is the low-order bit.

When consecutive octets are used to represent a binary number, the lower octet number has the most significant value.

Any subnetwork supporting this protocol is required to state in its specification the way octets are transferred, using the terms "most significant bit" and "least significant bit." The PDUs of this protocol are defined using the terms "most significant bit" and "least significant bit."

Note:

When the encoding of a PDU is represented using a diagram in this section, the following representation is used:

a) octets are shown with the lowest numbered octet to the left, higher number octets being further to the right;

b) within an octet, bits are shown with bit eight (8) to the left and bit one (1) to the right.

PDUs shall contain, in the following order:

1) the header, comprising:

a) the fixed part;

b) the address part;

c) the segmentation part, if present;

d) the options part, if present

and

2) the data field, if present.

This structure is illustrated below:

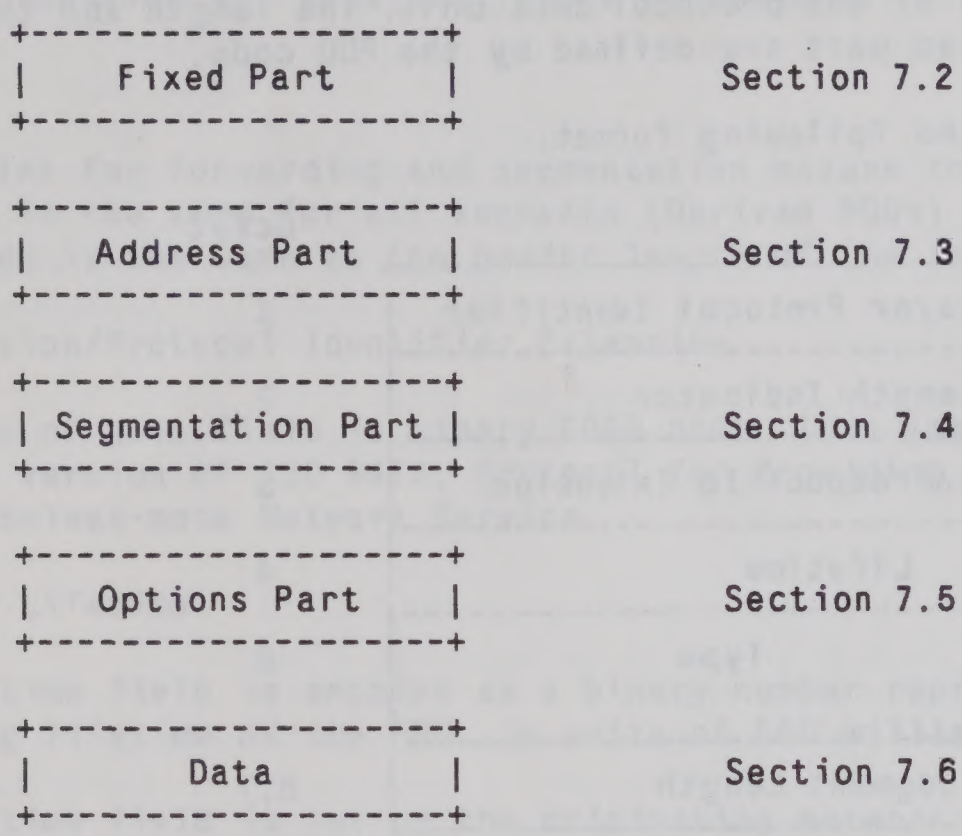


Figure 7-1. PDU Structure

7.2 Fixed Part

7.2.1 General

The fixed part contains frequently occuring parameters including the type code (DT or ER) of the protocol data unit. The length and the structure of the fixed part are defined by the PDU code.

The fixed part has the following format:

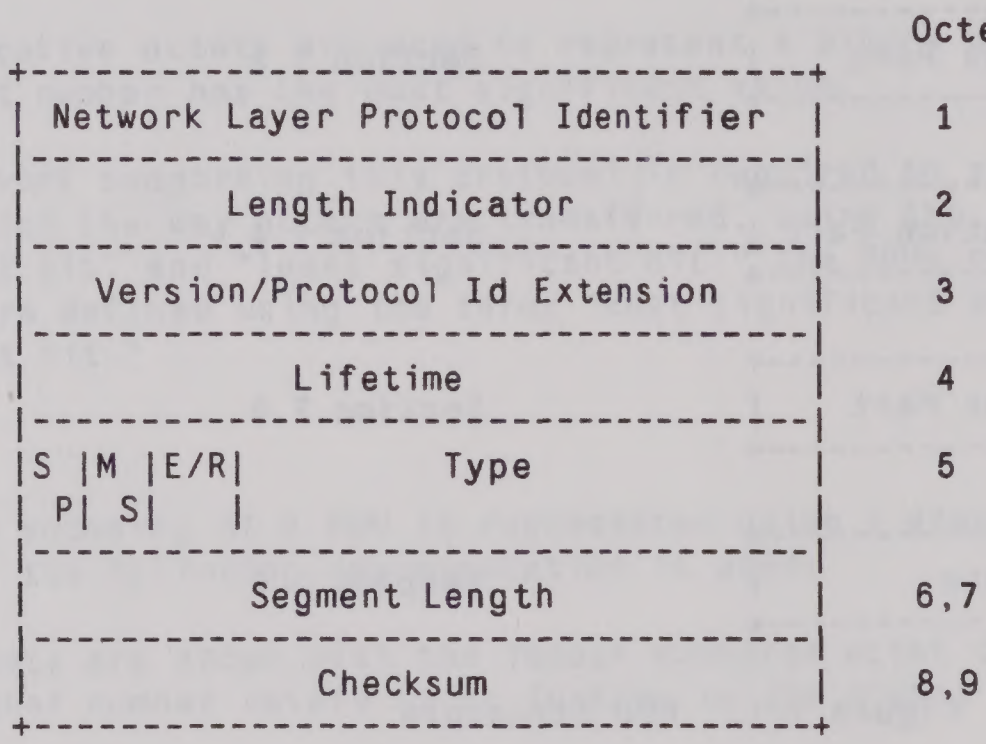


Figure 7-2. PDU Header--Fixed Part

7.2.2 Network Layer Protocol Identifier

The value of this field shall be binary 1000 0001. This field identifies this Network Layer Protocol as ISO 8473, Protocol for Providing the Connectionless-mode Network Service.

7.2.3 Length Indicator

The length is indicated by a binary number, with a maximum value of 254 (1111 1110). The length indicated is the length in octets of the header, as described in Section 7.1, Structure. The value 255 (1111 1111) is reserved for possible future extensions.

Note:

The rules for forwarding and segmentation ensure that the header length is the same for all segments (Derived PDUs) of the Initial PDU, and is the same as the header length of the Initial PDU.

7.2.4 Version/Protocol Identifier Extension

The value of this field is binary 0000 0001. This Identifies a standard version of ISO 8473, Protocol for Providing the Connectionless-mode Network Service.

7.2.5 PDU Lifetime

The Lifetime field is encoded as a binary number representing the remaining lifetime of the PDU, in units of 500 milliseconds.

The Lifetime field is set by the originating network-entity, and is decremented by every network-entity which processes the PDU. The PDU shall be discarded if the value of the field reaches zero.

When a network-entity processes a PDU, it decrements the Lifetime by at least one. The Lifetime shall be decremented by more than one if the sum of:

1) the transit delay in the subnetwork from which the PDU was received; and

2) the delay within the system processing the PDU

exceeds or is estimated to exceed 500 milliseconds. In this case, the lifetime field should be decremented by one for each additional 500 milliseconds of delay. The determination of delay need not be precise, but where error exists the value used shall be an overestimate, not an underestimate.

If the Lifetime reaches a value of zero before the PDU is delivered to the destination, the PDU shall be discarded. The Error Reporting function shall be invoked, as described in Section 6.10, Error Reporting Function, and may result in the generation of an ER PDU. It is a local matter whether the destination network-entity performs the Lifetime Control function.

When the Segmentation function is applied to a PDU, the Lifetime field is copied into all of the Derived PDUs.

7.2.6 Flags

7.2.6.1 Segmentation Permitted and More Segments Flags

The Segmentation Permitted flag determines whether segmentation is permitted. A value of one indicates that segmentation is permitted. A value of zero indicates that the non-segmenting protocol subset is employed. Where this is the case, the segmentation part of the PDU header is not present, and the Segment Length field serves as the Total Length field.

The More Segments flag indicates whether the data segment in this PDU contains (as its last octet) the last octet of the User Data in the NSDU. When the More Segments flag is set to one (1) then segmentation has taken place and the last octet of the NSDU is not contained in this PDU. The More Segments flag cannot be set to one (1) if the Segmentation Permitted flag is not set to one (1).

When the More Segments flag is set to zero (0) the last octet of the Data Part of the PDU is the last octet of the NSDU.

7.2.6.2 Error Report FTag

When the Error Report flag is set to one, the rules in Section 6.10 are used to determine whether to generate an Error Report PDU upon discard of the PDU.

When the Error Report flag is set to zero, discard of the PDU will not cause the generation of an Error Report PDU.

7.2.7 Type Code

The Type code field identifies the type of the protocol data unit. Allowed values are given in Table 7-1:

Bits	5 4 3 2 1
DT PDU	1 1 1 0 0
ER PDU	0 0 0 0 1

Table 7-1. Valid PDU Types

7.2.8 PDU Segment Length

The Segment Length field specifies the entire length of the PDU segment including both header and data, if present. When the full protocol is employed and a PDU is not segmented, then the value of this field is identical to the value of the Total Length field located in the Segmentation Part of the header.

When the Non-segmenting protocol subset is employed, no segmentation part is present in the header. In this subset, the Segment Length field serves as the Total Length field of the header (see Section 7.4.3).

7.2.9 PDU Checksum

The checksum is computed on the entire PDU header. This includes the segmentation and options parts, if present. A checksum value of zero is reserved to indicate that the checksum is to be ignored. The operation of the PDU Header Error Detection function ensures that the value zero does not represent a valid checksum. A non-zero value indicates that the checksum must be processed or the PDU must be discarded.

7.3 Address Part

7.3.1 General

Address parameters are distinguished by their location, immediately following the fixed part of the PDU header. The address part is illustrated below:

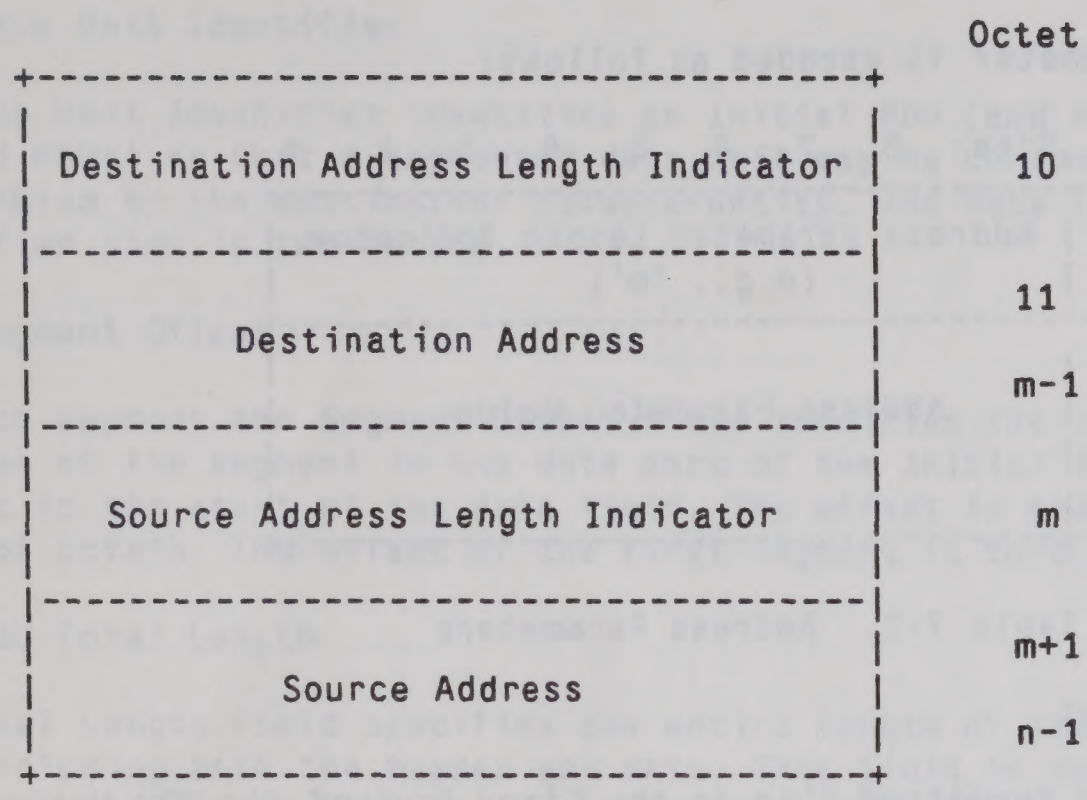


Figure 7-3. PDU header--Address Part

7.3.1.1 Destination and Source Address Information

The Destination and Source addresses are Network Service Access Point addresses as defined in ISO 8348/DAD2, Addendum to the Network Service Definition Covering Network Layer Addressing.

The Destination and Source Address information is of variable length.

The Destination Address Length Indicator field specifies the length of the Destination Address in number of octets. The Destination Address field follows the Destination Address Length Indicator field. The Source Address Length Indicator field specifies the length of the Source Address in number of octets. The Source Address Length Indicator field follows the Destination Address field. The Source Address field follows the Source Address Length Indicator field.

Each address parameter is encoded as follows:

Bits	8	7	6	5	4	3	2	1
Octet n	Address parameter Length Indicator (e.g., 'm')							
Octets n+1 thru n+m	Address Parameter Value							

Table 7-2. Address Parameters

7.4 Segmentation Part

If the Segmentation Permitted Flag in the Fixed Part of the PDU Header (Octet 4, Bit 8) is set to one, the segmentation part of the header, illustrated below, must be present:

	Octet
Data Unit Identifier	n,n+1
Segment Offset	n+2,n+3
Total Length	n+4,n+5

Figure 7-4. PDU Header--Segmentation Part

Where the "Segmentation Permitted" flag is set to zero, the nonsegmenting protocol subset is in use.

7.4.1 Data Unit Identifier

The Data Unit Identifier identifies an Initial PDU (and hence, its Derived PDUs) so that a segmented data unit may be correctly reassembled by the destination network-entity. The Data Unit Identifier size is two octets.

7.4.2 Segment Offset

For each segment the Segment Offset field specifies the relative position of the segment in the data part of the Initial PDU with respect to the start of the data field. The offset is measured in units of octets. The offset of the first segment is zero.

7.4.3 PDU Total Length

The Total Length field specifies the entire length of the Initial PDU, including both the header and data. This field is not changed in any segment (Derived PDU) for the lifetime of the PDU.

7.5 Options Part

7.5.1 General

The options part is used to convey optional parameters. If the options part is present, it contains one or more parameters. The number of parameters that may be contained in the options part is indicated by the length of the options part which is:

PDU Header Length - (length of fixed part +
length of address part +
length of segmentation part).

The options part of the PDU header is illustrated below:

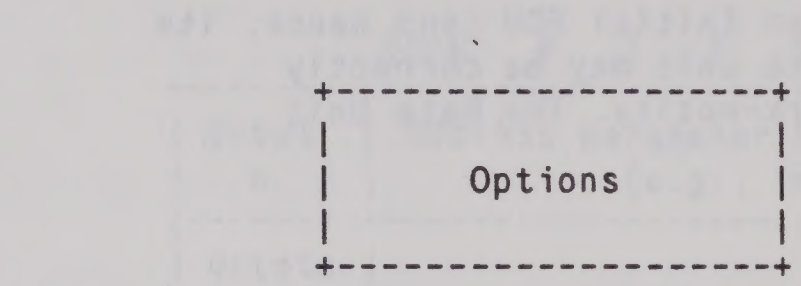


Figure 7-5. PDU Header--Options Part

Each parameter contained within the options part of the PDU header is encoded as follows:

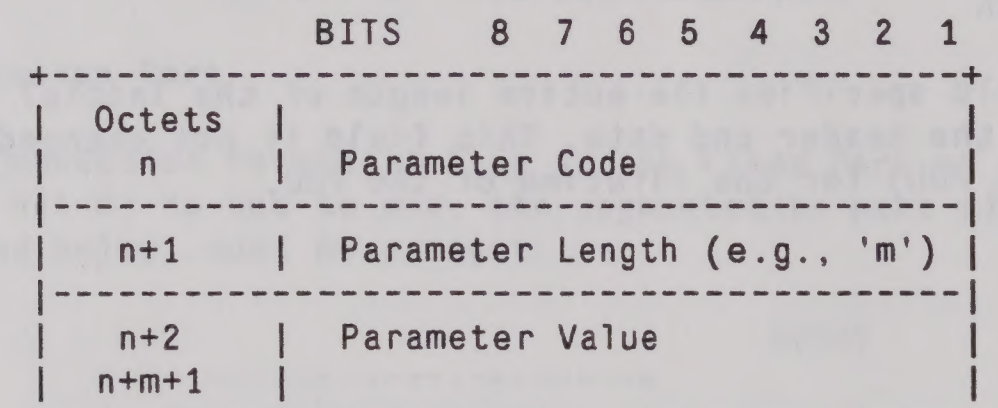


Table 7-3. Encoding of Parameters

The parameter code field is coded in binary and, without extensions, provides a maximum number of 255 different parameters. However, as noted below, bits 8 and 7 cannot take every possible value, so the practical maximum number of different parameters is less. A parameter code of 255 (binary 1111 1111) is reserved for possible extensions of the parameter code.

The parameter length field indicates the length, in octets, of the parameter value field. The length is indicated by a binary number, 'm', with a theoretical maximum value of 255. The practical maximum value of 'm' is lower. For example, in the case of a single parameter contained within the options part, two octets are required for the parameter code and the parameter length indication itself. Thus, the value of 'm' is limited to:

253 - (length of fixed part +
length of address part +
length of segmentation part).

For each succeeding parameter the maximum value of 'm' decreases.

The parameter value field contains the value of the parameter identified in the parameter code field.

No parameter codes use bits 8 and 7 with the value 00.

Implementations shall accept the parameters defined in the options part in any order. Duplication of options (where detected) is not permitted. Receipt of a PDU with an option duplicated should be treated as a protocol error. The rules governing the treatment of protocol errors are described in Section 6.10, Error Reporting Function.

The following parameters are permitted in the options part.

7.5.2 Padding

The padding parameter is used to lengthen the PDU header to a convenient size (See Section 6.12).

Parameter Code: 1100 1100
Parameter Length: variable
Parameter Value: any value is allowed

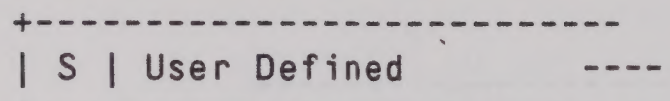
7.5.3 Security

This parameter is user defined.

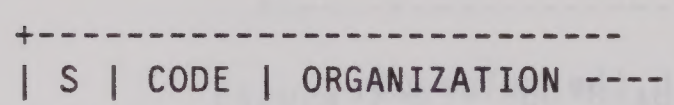
Parameter Code: 1100 0101
Parameter Length: variable
Parameter Value:

High order bit of first octet is Security Domain bit, S, to be interpreted as follows:

S=0



S=1



where

CODE = This field contains a geographic or non-geographic code to which the option applies.

ORGANIZATION = This is a further subdivision of the CODE field and is determined by an administrator of the geographic or non-geographic domain identified by the value of CODE.

7.5.4 Source Routing

The source routing parameter specifies, either completely or partially, the route to be taken from Source Network Address to Destination Network Address.

Parameter Code:	1100 1000
Parameter Length:	variable
Parameter Value:	2 octet control information succeeded by a concatenation of ordered address fields (ordered from source to destination)

The first octet of the parameter value is the type code. This has the following significance.

0000 0001	complete source routing
0000 0000	partial source routing

<all other values reserved>

The second octet indicates the octet offset of the next address to be processed in the list. A value of three (3) indicates that the next address begins immediately after this control octet. Successive octets are indicated by correspondingly larger values of this indicator.

The third octet begins the intermediate-system address list. The address list consists of variable length address fields. The first octet of each address field identifies the length of the address which comprises the remainder of the address field.

7.5.5 Recording of Route

The recording of route parameter identifies the route of intermediate systems traversed by the PDU.

Parameter Code:	1100 1011
Parameter Length:	variable
Parameter Value:	two octets control information succeeded by a concatenation of ordered addresses

The first octet is used to indicate that the recording of route has been terminated owing to lack of space in the option. It has the following significance:

0000 0000	Recording of Route still in progress
1111 1111	Recording of Route terminated

<all other values reserved>

The second octet identifies the next octet which may be used to record an address. It is encoded relative to the start of the parameter, such that a value of three (3) indicates that the octet after this one is the next to be used.

The third octet begins the address list. The address list consists of variable length address fields. The first octet of each address field identifies the length of the address which comprises the remainder of the field. Address fields are always added to the beginning of the address list; i.e., the most recently added field will begin in the third octet of the parameter value.

7.5.6 Quality of Service Maintenance

The Quality of Service parameter conveys information about the quality of service requested by the originating Network Service user. At intermediate systems, Network Layer relay entities may (but are not required to) make use of this information as an aid in selecting a route when more than one route satisfying other routing criteria is available and the available routes are known to differ with respect to Quality of Service (see Section 6.16).

Parameter Code:	1100 0011
Parameter Length:	one octet
Parameter Value:	Bit 8: transit delay vs. cost
	Bit 7: residual error probability vs. transit delay
	Bit 6: residual error probability vs. cost
	Bits 5 thru 0 are not specified.

Bit 8 is set to one indicates that where possible, routing decision should favor low transit delay over low cost. A value of 0 indicates that routing decisions should favor low cost over low transit delay.

Bit 7 set to one indicates that where possible, routing decisions should favor low residual error probability over low transit delay. A value of zero indicates that routing decisions should favor low transit delay over low residual error probability.

Bit 6 is set to one indicates that where possible, routing decisions should favor low residual error probability over low cost. A value of 0 indicates that routing decisions should favor low cost over low residual error probability.

7.6 Priority

The priority parameter carries the relative priority of the protocol data unit. Intermediate systems that support this option should make use of this information in routing and in ordering PDUs for transmission.

Parameter Code:	1100 1100
Parameter Length:	one octet
Parameter Value:	0000 0000 - Normal (Default)
	thru
	0000 1111 - Highest

The values 0000 0001 through 0000 1111 are to be used for higher priority protocol data units. If an intermediate system does not support this option then all PDUs shall be treated as if the field had the value 0000 0000.

7.7 Data Part

The Data part of the PDU is structured as an ordered multiple of octets, which is identical to the same ordered multiple of octets specified for the NS_Userdata parameter of the N_UNITDATA Request and Indication primitives. The data field is illustrated below:

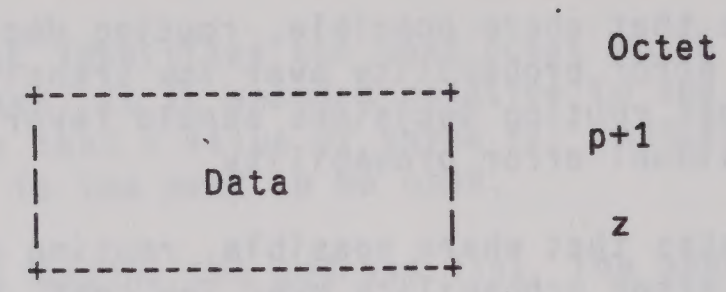


Figure 7-6. PDU header--Data Field

7.8 Data (DT) PDU

7.8.1 Structure

The DT PDU has the following format:

	Octet
Network Layer Protocol Identifier	1
Length Indicator	2
Version/Protocol Id Extension	3
Lifetime	4
SP | MS | E/R | Type	5
Segment Length	6,7
Checksum	8,9
Destination Address Length Indicator	10
Destination Address	11 through m-1
Source Address Length Indicator	m
Source Address	m+1 through n-1
Data Unit Identifier	n,n+1
Segment Offset	n+2,n+3
Total Length	n+4,n+5
Options	n+6 through p
Data	p+1 through z

Figure 7-7. PDU Header Format

7.8.1.1 Fixed Part

1) Network Layer Protocol Identifier	See Section 7.2.2.
2) Length Indicator	See Section 7.2.3.
3) Version/Protocol Id Extension	See Section 7.2.4.
4) Lifetime	See Section 7.2.5.
5) SP, MS, E/R	See Section 7.2.6.
6) Type Code	See Section 7.2.7.
7) Segment Length	See Section 7.2.8.
8) Checksum	See Section 7.2.9.

7.8.1.2 Addresses

See Section 7.3.

7.8.1.3 Segmentation

See Section 7.4.

7.8.1.4 Options

See Section 7.5.

7.8.1.5 Data

See Section 7.7.

7.9 Inactive Network Layer Protocol

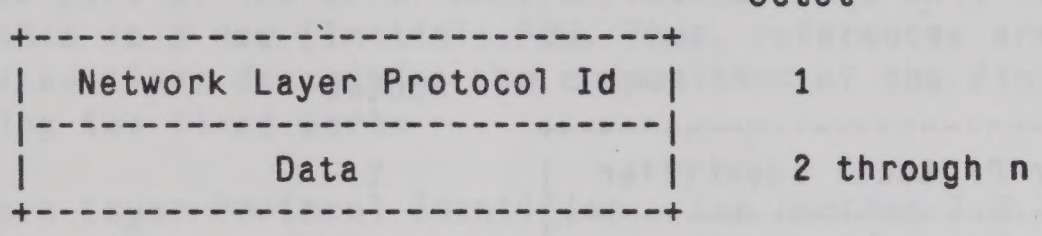


Figure 7-9. Inactive Network Layer Protocol

7.9.1 Network Layer Protocol Id

The value of the Network Layer Protocol Identifier field is binary zero (0000 0000).

7.9.2 Data Field

See Section 7.7.

The length of the NS_Userdata parameter is constrained to be less than or equal to the value of the length of the SN_Userdata parameter minus one.

7.10 Error Report PDU (ER)

7.10.1 Structure

	Octet
Network Layer Protocol Identifier	1
Length Indicator	2
Version/Protocol Id Extension	3
Lifetime	4
SP | MS | E/R | Type	5
Segment Length	6,7
Checksum	8,9
Destination Address Length Indicator	10
Destination Address	10 through m-1
Source Address Length Indicator	m
Source Address	m+1 through n-1
Data Unit Identifier	n,n+1
Segment Offset	n+2,n+3
Total Length	n+4,n+5
Options	n+6 through p-1
Reason for Discard	p through q-1
Error Report Data Field	z

Figure 7-10. Error Report PDU

7.10.1.1 Fixed Part

The fixed part of the Error Report Protocol Data Unit is set as though this is a new (Initial) PDU. Thus, references are provided to precious sections describing the composition of the fields comprising the fixed part:

1) Network Layer Protocol Identifier	See Section 7.2.2.
2) Length Indicator	See Section 7.2.3.
3) Version/Protocol Id Extension	See Section 7.2.4.
4) Lifetime	See Section 7.2.5.
5) SP, MS, E/R	See Section 7.2.6.
6) Type Code	See Section 7.2.7.
7) Segment Length	See Section 7.2.8.
8) Checksum	See Section 7.2.9.

7.10.1.2 Addresses

See Section 7.3.

The Destination Address specifies the original source of the discarded PDU. The Source Address specifies the intermediate system or end system network-entity initiating the Error Report PDU.

7.10.1.3 Segmentation

See Section 7.4.

7.10.1.4 Options

See Section 7.5.

7.10.1.5 Reason for Discard

This parameter is only valid for the Error Report PDU. It provides a report on the discarded protocol data unit.

Parameter Code:

1100 0001

Parameter Length:

two octets
type of error encoded in binary:

```
0000 0000:  Reason not specified.
0000 0001:  Protocol Procedure Error.
            other than below:
0000 0010:  Incorrect checksum.
0000 0011:  PDU discarded due to congestion.
0000 0100:  Header syntax error (header cannot
            be parsed).
0000 0101:  Segmentation is needed but is not
            permitted.

1000 xxxx:  Addressing Error:
            0000 0000:  Destination Address
                        Unreachable.
            1000 0001:  Destination Address
                        Unknown.

1001 xxxx:  Source Routing Error:
            1001 0000:  Unspecified Source
                        Routing error.
            1001 0001:  Syntax error in Source
                        Routing field.
            1001 0010:  Unknown Address in
                        Source Routing field.
            1001 0011:  Path not acceptable.
```

1010 xxxx: Lifetime Expiration:
- 1010 0000: Lifetime expired while data unit in transit.
- 1010 0001: Lifetime expired during reassembly.

1011 xxxx: PDU discarded due to unsupported option:
- 1011 0000: unsupported option not specified.
- 1011 0001: unsupported padding option.
- 1011 0010: unsupported security option.
- 1011 0011: unsupported source routing option.
- 1011 0100: unsupported recording of route option.
- 1011 0101: unsupported QoS Maintenance option.

The second octet contains a pointer to the field in the associated discarded PDU which caused the error. If no one particular field can be associated with the error, then this field contains the value of zero.

7.10.1.6 Error Report Data Field

This field provides all or a portion of the discarded PDU. The octets comprising this field contain the rejected or discarded PDU up to and including the octet which caused the rejection/discard.

8 FORMAL DESCRIPTION

The operation of the protocol is modelled as a finite state automaton governed by a state variable with three values. The behavior of the automaton is defined with respect to individual independent Protocol Data Units. A transition of the automaton is prompted by the occurrence of an atomic event at one of three interfaces:

1) an interface to the Transport Layer, defined by the service primitives of the Addendum to the Network Service Definition Covering Connectionless-mode Transmission;

2) an interface to the subnetwork service provider, defined by the SN_UNITDATA primitive of Section 5.5 of this Standard;

3) an interface to an implementation-dependent timer function defined by the TIMER primitives described in Section 5.6 of this Standard.

In addition, a transition of the automaton may be prompted by the occurrence of a condition of the automaton.

The atomic events are defined in Section 8.2. The occurrence of an atomic event is not in itself sufficient to cause a transition to take place; other conditions, called "enabling conditions" may also have to be met before a particular transition can take place. Enabling conditions are boolean expressions that depend on the values of parameters associated with the corresponding atomic event (that is, the parameters of some primitive), and on the values of locally maintained variables.

More than one enabling condition -- and therefore, more than one possible transition -- may be associated with a single atomic event. In every such case, the enabling conditions are mutually exclusive, so that for any given combination of atomic event and parameter values, only one state transition can take place.

Associated with each transition is an action, or "output." Actions consist of changes to the values of local variables and the sequential performance of zero or more functions. The operation of the finite state automaton is completely specified in Section 8.3 by defining the action associated with every possible transition.

8.1 Values of the State Variable

The protocol state variable has three values:

1) INITIAL — The automaton is created in the INITIAL state. No transition may carry the automaton into the INITIAL state.

2) REASSEMBLING — The automaton is in the REASSEMBLING state for the period in which it is assembling PDU segments into a complete PDU.

3) CLOSED — The final state of the automaton is the CLOSED state. When the automaton enters the CLOSED state it ceases to exist.

8.2 Atomic Events

An atomic event is the transfer of a unit of information across an interface. The description of an atomic event specifies a primitive (such as an N_UNITDATA.Request), and the service boundary at which it is invoked (such as the Network Service boundary). The direction of information flow across the boundary is implied by the definition of each of the primitives.

8.2.1 N.UNITDATA_request and N.UNITDATA_indication

The N.UNITDATA_request and N.UNITDATA_indication atomic events occur at the Network Service boundary. They are defined by the Addendum to the Network Service Definition Covering Connectionless Data Transmission (ISO 8348/DAD1).

```
N.UNITDATA_request     (NS Source_Address,
                        NS_Destination_Address,
                        NS_Quality_of_Service,
                        NS_Userdata)

N.UNITDATA_indication (NS_Source_Address,
                        NS_Destination_Address,
                        NS_Quality_of_Service, NS_Userdata)
```

The parameters of the N.UNITDATA_request and N.UNITDATA_indication are collectively referred to as Network Service Data Unit (NSDUs).

8.2.2 SN.UNITDATA_request and SN.UNITDATA_indication

The SN.UNITDATA_request and SN.UNITDATA_indication atomic events occur at the interface between the Protocol described herein and a subnetwork service provider. They are defined in Section 5.5 of this Standard.

```
SN.UNITDATA_request     (SN_Source_Address,
                         SN_Destination_Address,
                         SN_Quality_of_Service,
                         SN_Userdata)

SN.UNITDATA_indication (SN_Source_Address,
                         SN_Destination_Address,
                         SN_Quality_of_Service,
                         SN_Userdata)
```

The parameters of the SN_UNITDATA request and SN_UNITDATA Indication are collectively referred to as Subnetwork Service Data Units (SNSDUs).

The value of the SN_Userdata parameter may represent an Initial PDU or a Derived PDU.

8.2.3 TIMER Atomic Events

The TIMER atomic events occur at the interface between the Protocol described herein and its local environment. They are defined in Section 5.6 of this Standard.

```
S.TIMER_request  (Time,
                  Name,
                  Subscript)

S.TIMER_cancel   (Name
                  Subscript)

S.TIMER_response (Name,
                  Subscript)
```

8.3 Operation of the Finite State Automation

The operation of the automaton is defined by use of the formal description technique and notation specified in ISO/TC97/SC16 N1347. This technique is based on an extended finite state transition model and the Pascal programming language. The technique makes use of strong variable typing to reduce ambiguity in interpretation of the specification.

This specification formally specifies an abstract machine which provides a single instance of the Connectionless-Mode Network Service by use of the Protocol For Providing the Connectionless-Mode Network Service. It should be emphasized that this formal specification does not in any way constrain the internal operation or design of any actual implementation. For example, it is not required that the program segments contained in the state transitions will actually appear as part of an actual implementation. A formal protocol specification is useful in that it goes as far as possible to eliminate any degree of ambiguity or vagueness in the specification of a protocol standard.

The formal specification contained here specifies the behavior of a single finite-state machine, which provides the protocol

behavior corresponding to a single independent service request. It is expected that any actual implementation will be able to handle behavior corresponding to many simultaneous finite state machines.

8.3.1 Type and Constant Definitions

```
const

 ZERO  = 0;
 max_user_data = 64512;

type

 NSAP_addr_type  = ...;

  { NSAP_addr_type defines the data type for NSAP addresses, as
  passed across the Network Service Boundary. }

 NPAI_addr_type  = ...;

  { NPAI_addr_type defines the data type for the addresses carried in
  PDUs. }

 SN_addr_type    = ...;

  { SN_addr_type defines the data type for addresses in the
  underlying service used by this protocol. }

 quality_of_service_type = ...;

  { Quality_of_service_type defines the data type for the QOS
  parameter passed across the Network Service boundary. }

 SN_QOS_type     = ...;

  { SN_QOS_type defines the data type for the QOS parameter, if any,
  passed to the underlying service used by this protocol. }

 data_type       = ...;

  { Data_type defines the data type for user data. Conceptually this
  is equivalent to a variable length binary string. }

 buffer_type     = ...;

  { Buffer_type defines the data type for the memory resources used
  in sending and receiving of user data.  This provides capabilities
  required for segmentation and reassembly. }
```

```
timer_name_type = (lifetime_timer);
timer_data_type = ...;

network_layer_protocol_id_type = (ISO_8473_protocol_id);
version_id_type  = (version1);
pdu_tp_type      = (DT, ER);

options_type     = ...;

 { Options_type defines the data type used to store the options part
 of the PDU header. }

subnet_id_type   = ...;

 { The subnet_id_type defines the data type used to locally identify
 a particular underlying service used by this protocol.  In general
 there may be multiple underlying subnetwork (or data link)
 services. }

error_type       = (NO_ERROR,
                    TOO_MUCH_USER_DATA,
                    PROTOCOL_PROCEDURE_ERROR,
                    INCORRECT_CHECKSUM, CONGESTION,
                    SYNTAX_ERROR,
                    SEG_NEEDED_AND_NOT_PERMITTED,
                    DESTINATION_UNREACHABLE,
                    DESTINATION_UNKNOWN,
                    UNSPECIFIED_SRC_ROUTING_ERROR,
                    SYNTAX_ERROR_IN_SRC_ROUTING,
                    UNKNOWN_ADDRESS_IN_SRC_ROUTING,
                    PATH_NOT_ACCEPTABLE_IN_SRC_ROUTING,
                    LIFETIME_EXPIRED_IN_TRANSIT,
                    LIFETIME_EXPIRED_IN_REASSEMBLY,
                    UNSUPPORTED_OPTION_NOT_SPECIFIED,
                    UNSUPPORTED_PADDING_OPTION,
                    UNSUPPORTED_SECURITY_OPTION,
                    UNSUPPORTED_SRC_ROUTING_OPTION,
                    UNSUPPORTED_RECORDING_OF_ROUTE_OPTION,
                    UNSUPPORTED_QOS_MAINTENANCE_OPTION);
```

```
nsdu_type = record
                da    : NSAP_addr_type;
                sa    : NSAP_addr_type;
                qos   : quality_of_service_type;
                data  : data_type;
            end;

pdu_type = record
                nlp_id   : network_layer_protocol_id_type;
                hli      : integer;
                vp_id    : version_id_type; lifetime : integer;
                sp       : boolean;
                ms       : boolean;
                er_flag  : boolean;
                pdu_tp   : pdu_tp_type;
                seg_len  : integer;
                checksum : integer;
                da_len   : integer;
                da       : NPAI_addr_type;
                sa_len   : integer;
                sa       : NPAI_addr_type;
                du_id    : optional integer;
                so       : optional integer;
                tot_len  : optional integer;
                   { du_id, so, and tot_len are present
                    only if sp has the value TRUE. }
              options  : options_type;
              data     : data_type;
        end;
```

```
route_result_type =
          record

        subnet_id    : subnet_id_type;
        sn_da        : SN_addr_type;
        sn_sa        : SN_addr_type;
        segment_size : integer;
      end;
```

8.3.2 Interface Definitions

```
channel Network_access_point (User, Provider);

 by User:
     UNITDATA_request
        (NS_Destination_address : NSAP_addr_type;
         NS_Source_address      : NSAP_addr_type;
         NS_Quality_of_Service  : quality_of_service_type;
         NS_Userdata            : data_type);

 by Provider:
     UNITDATA_indication
        (NS_Destination_address : NSAP_addr_type;
         NS_Source_address      : NSAP_addr_type;
         NS_Quality_of_Service  : quality_of_service_type;
         NS_Userdata            : data_type);

channel Subnetwork_access_point (User, Provider);

 by User:
     UNITDATA_request
        (SN_Destination_address : SN_addr_type;
         SN_Source_address      : SN_addr_type;
         SN_Quality_of_Service  : SN_QOS_type;
         SN_Userdata            : pdu_type);

 by Provider:
     UNITDATA_indication
        (SN_Destination_address : SN_addr_type;
         SN_Source_address      : SN_addr_type;
         SN_Quality_of_Service  : SN_QOS_type;
         SN_Userdata            : pdu_type);

channel System_access_point (User, Provider);

 by User:
     TIMER_request
        (Time      : integer;
         Name      : timer_name_type;
         Subscript : integer);
```

```
    TIMER_cancel

         (Name      : timer_name_type;
          Subscript : integer);

by Provider:
    TIMER_indication
       (Name      : timer_name_type;
        Subscript : integer);
```

8.3.3 Formal Machine Definition

```
module Connectionless_Network_Protocol_Machine
     (N:  Network_access_point (Provider) common queue;
      SN: array [subnet_id_type] of Subnetwork_access_point
                                          (User) common queue;
      S:  System_access_point (User) individual queue );

var
    nsdu    : nsdu_type;
    pdu     : pdu_type;
    rcv_buf : buffer_type;

state : (INITIAL, REASSEMBLING, CLOSED);
```

```
procedure send_error_report (error : error_type;
                             pdu   : pdu_type);

 var
     er_pdu : pdu_type;

 begin
  if (pdu.er_flag) then
   begin
    er_pdu.nlp_id    := ISO_8473_protocol_id;
    er_pdu.vp_id     := version1;
    er_pdu.lifetime  := get_er_lifetime(pdu.sa);
    er_pdu.sp        := get_er_seg_per(pdu);
    er_pdu.ms        := FALSE;
    er_pdu.er_flag   := FALSE;
    er_pdu.pdu_tp    := ER;
    er_pdu.da_len    := pdu.sa_len;
    er_pdu.da        := pdu.sa;
    er_pdu.sa_len    := get_local_NPAI_addr_len;
    er_pdu.sa        := get_local_NPAI_addr;
    er_pdu.options   := get_er_options
                        (error,
                        er_pdu.da,
                        pdu.options);
    er_pdu.hli       := get_header_length
                        (er_pdu.da_len, er_pdu.sa_len,
                         er_pdu.sp,
                         er_pdu.options);
    er_pdu.data      := get_er_data_field(error, pdu);
    if (er_pdu.sp) then
                        begin
                           er_pdu.du_id   :=
                           get_data_unit_id(er_pdu.da);
                           er_pdu.so      := ZERO;
                           er_pdu.tot_len := er_pdu.hli +
                           size(er_pdu.data);
                        end;
```

```
        if (NPAI_addr_local(er_pdu.da))
                        then
                           post_error_report(er_pdu)
                        else
                           send_pdu(er_pdu);
      end;
    end;
```

```
procedure send_pdu (pdu : pdu_type);

 var

  rte_result   : route_result_type;
  error_code   : error_type;
  send_buf     : buffer_type;
  data_maxsize : integer;
  more_seg     : boolean;
  sn_qos       : SN_QOS_type;

 begin

  send_buf := make_buffer(pdu.data);
  more_seg := pdu.ms;

  repeat

   begin

    error_code := check_parameters
                  (pdu.hli,
                   pdu.sp,
                   pdu.da,
                  pdu.options,
                  size(pdu.data));

  if (error_code = NO_ERROR) then

                    begin

                       rte_result := route(pdu.hli,
                                           pdu.sp,
                                           pdu.da,
                                           pdu.options,
                                           size(pdu.data));

                       data_maxsize := rte_result.segment_size -
                       pdu.hli;
                       pdu.data     := extract(send_buf,
                       data_maxsize);
                       pdu.seg_len  := pdu.hli + size(pdu.data);

                       if (size(send_buf) = ZERO) then
                           pdu.ms   := more_seg
                       else
                           pdu.ms   := TRUE;
```

```
                    pdu.checksum := get_checksum(pdu);
                    sn_qos       := get_sn_qos
                    (rte_result.subnet_id,
                                              pdu.options);

                    out SN[rte_result.subnet_id].UNITDATA_request
                              (rte_result.sn_da,
                               rte_result.sn_sa,
                               sn_qos,
                               pdu);

                    pdu.so := pdu.so + data_maxsize;

                end

else if (error_code = CONGESTION) then

                begin

                 if (send_er_on_congestion (pdu)) then
                     send_error_report(CONGESTION, pdu);

                end

                 else

                              send_error_report(error_code, pdu);

                 end;

                until (size_buf(data_buf) = ZERO) or
                      (error_code <> NO_ERROR);

end;
```

```
procedure allocate_reassembly_resources
        (pdu_tot_len : integer);
primitive;

 { This procedure allocates resources required for reassembly of a
 PDU of the specified total length.  If this requires discarding of a
 PDU in which the ER flag is set, then an error report is returned to
 the source of the discarded data unit. }

function check_parameters
     (hli     : integer;
      sp      : boolean;
      da      : NPAI_addr_type;
      options : options_type;
      datalen : integer) : error_type;
primitive;

 { This function examines various parameters associated with a PDU,
 to determine whether forwarding of the PDU can continue.  If a
 result of NO_ERROR is returned, then the primitive route can be
 called to specify the route and segment size.  Otherwise this
 function specifies the reason that an error has occurred. }

function data_unit_complete
     (buf : buffer_type) : boolean;
primitive;

 { This function returns a boolean value specifying whether the PDU
 stored in the specified buffer has been completely received. }
```

```
function elapsed_time : integer;
primitive;

 { This function returns an estimate of the time elapsed, in 500
 microsecond increments, since the PDU was transmitted by the
 previous peer network entity.  This estimate includes both time
 spent in transit, and any time to be spent in buffers within the
 local system.  Although this estimate need not be precise,
 overestimates are preferable to underestimates, as underestimating
 the time elapsed may defeat the intent of the lifetime function. }

procedure empty_buffer
     (buf : buffer_type);
primitive;

 { This procedure empties the specified buffer. }

function extract
     (buf    : buffer_type;
      amount : integer) : data_type;
primitive;

 { This function removes the specified amount of data from
 the specified buffer, and returns this data as the function
 value. }

procedure free_reassembly_resources;
primitive;

 { This procedure releases the resources that had been previously
 allocated by the procedure allocate_reassembly_resources. }

function get_checksum
     (pdu : pdu_type) : integer;
primitive;

 { This function returns the 16 bit integer value to be placed in the
 checksum field of the PDU.  If the checksum facility is not being
 used, then this function returns the value zero.  The algorithm for
 producing a correct checksum value is specified in Annex A. }

function get_data_unit_id
     (da : NPAI_addr_type) : integer;
primitive;

 { This function returns a data unit identifier which is unique for
 the specified destination address. }
```

```
function get_er_data_field
     (error : error_type;
      pdu   : pdu_type) : data_type;
primitive;

 { This function returns the correct data field for an error report,
 based on the information that the specified PDU is being discarded
 due to the specified error.  The data field of an error report must
 include the header of the discarded PDU, and may optionally contain
 additional user data. }

function get_er_flag
     (nsdu : nsdu_type) : boolean;
primitive;

 { This function returns a boolean value to be used as the error
 report flag in a PDU which transmits the specified nsdu.  If the PDU
 must be discarded at some future time, an error report can be
 returned only if this value is set to TRUE. }

function get_er_lifetime
     (da : NPAI_addr_type) : integer;
primitive;

 { This function returns the lifetime value to be used for an error
 report being sent to the specified destination address. }

function get_er_options
     (error   : error_type;
      da      : NPAI_addr_type;
      options : options_type) : options_type;
primitive;

 { This function returns the options field of an error report, based
 on the reason for discard, and the destination address and options
 field of the discarded PDU.  The options field contains the reason
 for discard option, and may contain other optional fields. }
```

```
function get_er_seg_per

     (pdu      : pdu_type) : boolean;
primitive;

 { This function returns the boolean value which will be used for the
 segmentation permitted flag of an error report. }

function get_header_len
     (da_len   : integer;
      sa_len   : integer;
      sp       : boolean;
      options  : options_type) : integer;
primitive;

 { This function returns the header length, in octets.  This depends
 upon the lengths of the source and destination addresses, whether
 the segmentation part of the header is present, and the length of
 the options part. }

function get_lifetime
     (da   : NSAP_addr_type;
      qos  : quality_of_service_type) : lifetime_type;
primitive;

 { This function returns the lifetime value to be used for a PDU,
 based upon the destination address and requested quality of service.
 }

function get_local_NPAI_addr : NPAI_addr_type;
primitive;

 { This functions returns the local address as used in the protocol
 header. }

function get_local_NPAI_addr_len : integer;
primitive;

 { This functions returns the length of the local address as used in
 the protocol header. }
```

```
function get_NPAI
     (addr : NSAP_addr_type) : NPAI_addr_type;
primitive;

 { This function returns the network address as used in the protocol
 header, or "Network Protocol Addressing Information", corresponding
 to the specified NSAP address. }

function get_NPAI_len
     (addr : NSAP_addr_type) : integer;
primitive;

 { This function returns the length of the network address
 corresponding to a specified NSAP address. }

function get_NSAP_addr
     (addr : NPAI_addr_type;
      len  : integer) : NSAP_addr_type;
primitive;

 { This function returns the NSAP address corresponding to the
 network protocol addressing information (as it appears in the
 protocol header) of the specified length. }

function get_options
     (da  : NSAP_addr_type;
      qos : quality_of_service_type) : options_type;
primitive;

 { This function returns the options field for a PDU, based on the
 requested destination address and quality of service. }

function get_seg_permitted
     (da : NSAP_addr_type;
      qos : quality_of_service_type) : boolean;
primitive;

 { This function returns the boolean value to be used in the
 segmentation permitted field of a PDU.  This value may depend upon
 the destination address, requested quality of service, and the
 length of the user data. }
```

```
function get_sn_qos
     (subnet_id : subnet_id_type;

      options   : options_type) : SN_QOS_type;
primitive;

 { This function returns the quality of service to be used on the
 specified subnetwork, in order to obtain the quality of service (if
 any) and other parameters requested in the options part of the PDU.
 }

function get_qos
     (options : options_type) : quality_of_service_type;
primitive;

 { This function determines, to the extent possible, the quality of
 service that was obtained for a particular PDU, based upon the
 quality of service and other information contained in the options
 part of the PDU header. }

function make_buffer
     (data : data_type) : buffer_type;
primitive;

 { This function places the specified data in a newly created buffer.
 The precise manner of handling buffers is implementation specific.
 This newly created buffer is returned as the function value. }

procedure merge_seg
     (buf   : buffer_type;
      so    : integer;
      data  : data_type);
primitive;

 { This procedure merges the specified data into the specified
 buffer, based on the specified segment offset of the data. }

function NPAI_addr_local
     (addr : NPAI_addr_type) : boolean;
primitive;

 { This function returns the boolean value TRUE only if the specified
 network protocol addressing information specifies a local address. }
```

```
function NSAP_addr_local
     (addr : NSAP_addr_type) : boolean;
primitive;

 { This function returns the boolean value TRUE only if the specified
 NSAP address specifies a local address. }

procedure post_error_report
     (er_pdu : pdu_type);
primitive;

 { This procedure posts the specified error report (ER) type PDU to
 the appropriate local entity that handles error reports. }

function route
     (hli     : integer;
      sp      : boolean;
      da      : NPAI_addr_type;
      options : options_type;
      datalen : integer) : route_result_type;
primitive;

 { This function determines the route to be followed by a PDU
 segment, as well as the segment size. Note that in general, the
 segment size and route may be mutually dependent. This
 determination is made on the basis of the header length, the
 segmentation permitted flag, the destination address, several
 parameters (such as source routing) contained in the options part of
 the PDU header, and the length of data. This function returns a
 structure that specifies the subnetwork on which the segment should
 be transmitted, the source and destination addresses to be used on
 the subnetwork, and the segment size. This routine may only be
 called if the primitive function check_parameters has already
 determined that an error will not occur. }
```

```
function send_er_on_congestion
    (pdu : pdu_type) : boolean;
primitive;

 { This function returns the boolean value true if an error report
 should be sent when the indicated data unit is discarded due to
 congestion.  Note that if the value true is returned, then the
 er_flag field of the discarded data unit must still be checked
 before an error report can be sent. }

function size
    (data : data_type) : integer;
primitive;

 { This function returns the length, in octets, of the specified
 data. }

function size_buf
    (buf : buffer_type) : integer;
primitive;

 { This function returns the length, in octets, of the data contained
 in the specified buffer. }

initialize

 begin
     state to INITIAL;
 end;
```

```
trans  (* begin transitions *)

from INITIAL  to  CLOSED
when      N.UNITDATA_request
provided  not NSAP_addr_local(NS_Destination_Address)

begin
  nsdu.da   := NS_Destination_Address;
  nsdu.sa   := NS_Source_Address;
  nsdu.qos  := NS_Quality_o  _Service;
  nsdu.data := NS_Userdata;

  pdu.nlp_id   := ISO_8473_protocol_id;
  pdu.vp_id    := version1;
  pdu.lifetime := get_lifetime(nsdu.da, nsdu.qos);
  pdu.sp       := get_seg_permitted(nsdu.da, nsdu.qos);
  pdu.ms       := FALSE;
  pdu.er_flag  := get_er_flag(nsdu);
  pdu.pdu_tp   := DT;
  pdu.da_len   := get_NPAI_len(nsdu.da);
  pdu.da       := get_NPAI(nsdu.da);
  pdu.sa_len   := get_NPAI_len(nsdu.sa);
  pdu.sa       := get_NPAI(nsdu.sa);
  pdu.options  := get_options(nsdu.da, nsdu.qos);
  pdu.data     := nsdu.data;

  pdu.hli      := get_header_len(pdu.da_len,
                                 pdu.sa_len,
                                 pdu.sp,
                                 pdu.options);

  if (pdu.sp) then
       begin
         pdu.du_id    := get_data_unit_id(pdu.da);
         pdu.so       := ZERO;
         pdu.tot_len  := pdu.hli  +  size(pdu.data);
       end;

  if (size(pdu.data) > max_user_data) then
       send_error_report(TOO_MUCH_USER_DATA, pdu)
  else
       send_pdu(pdu);
end;
```

```
from INITIAL  to  CLOSED
when      N.UNITDATA_request
provided  NSAP_addr_local(NS_Destination_Address)

begin
  nsdu.da   := NS_Destination_Address;
  nsdu.sa   := NS_Source_Address;
  nsdu.qos  := NS_Quality_of_Service;
  nsdu.data := NS_Userdata;

  out N.UNITDATA_indication
      (nsdu.da, nsdu.sa, nsdu.qos, nsdu.data);

end;

from INITIAL  to  CLOSED
when      SN[subnet_id].UNITDATA_indication
provided  NPAI_addr_local(SN_Userdata.da)  and
          SN_Userdata.so       =  ZERO     and
          not  SN_Userdata.ms

begin
  pdu := SN_Userdata;
  if (pdu.pdu_tp = DT) then
      out N.UNITDATA_indication
         (get_NSAP_addr(pdu.da_len, pdu.da),
          get_NSAP_addr(pdu.sa_len, pdu.sa),
          get_qos(pdu.options),
          pdu.data)
  else
      post_error_report(pdu);

end;
```

```
from INITIAL  to  REASSEMBLING
when      SN[subnet_id].UNITDATA_indication
provided  NPAI_addr_local(SN_Userdata.da)    and
          ((SN_Userdata.so > ZERO) or (SN_Userdata.ms))

begin
  pdu := SN_Userdata;
  allocate_reassembly_resources(pdu.tot_len);
  empty_buffer(rcv_buf);

  merge_seg
     (rcv_buf,
      pdu.so,
      pdu.data);

  out S.TIMER_request
     (pdu.lifetime,
      lifetime_timer,
      ZERO);

end;

from INITIAL  to  CLOSED
when      SN[subnet_id].UNITDATA_indication
provided  not NPAI_addr_local(SN_Userdata.da)

begin
  pdu := SN_Userdata;

  if (pdu.lifetime > elapsed_time) then
    begin
      pdu.lifetime := pdu.lifetime - elapsed_time;
      send_pdu(pdu);
    end
else
    send_error_report(LIFETIME_EXPIRED, pdu);

end;
```

```
from REASSEMBLING  to  REASSEMBLING
when       SN[subnet_id].UNITDATA_indication
provided  (SN_Userdata.du_id    = pdu.du_id)    and
          (SN_Userdata.da_len   = pdu.da_len)   and
          (SN_Userdata.da       = pdu.da)       and
          (SN_Userdata.sa_len   = pdu.sa_len)   and
          (SN_Userdata.sa       = pdu.sa)

begin
  merge_seg
     (rcv_buf,
      SN_Userdata.so,
      SN_Userdata.data);

end;

from REASSEMBLING  to  CLOSED
provided  data_unit_complete(rcv_buf)
no delay

begin
  if (pdu.pdu_tp = DT) then
      out N.UNITDATA_indication
         (get_NSAP_addr(pdu.da_len, pdu.da),
          get_NSAP_addr(pdu.sa_len, pdu.sa),
          get_qos(pdu.options),
          extract (rcv_buf, size_buf(rcv_buf)))
 else
     post_error_report(pdu);
 out S.TIMER_cancel(lifetime_timer,ZERO);
 free_reassembly_resources;

end;

from REASSEMBLING  to  CLOSED
when       S.TIMER_indication

begin
  send_error_report(LIFETIME_EXPIRED, pdu);

end;
```

9 CONFORMANCE

For conformance to this International Standard, the ability to originate, manipulate, and receive PDUs in accordance with the full protocol (as opposed to the "non-segmenting" or "Inactive Network Layer Protocol" subsets) is required.

Additionally, the provision of the optional functions described in Section 6.17 and enumerated in Table 9-1 must meet the requirements described therein.

Additionally, conformance to the Standard requires adherence to the formal description of Section 8 and to the structure and encoding of PDUs of Section 7.

If and only if the above requirements are met is there conformance to this International Standard.

9.1 Provision of Functions for Conformance

The following table categorizes the functions in Section 6 with respect to the type of system providing the function:

Function	Send	Forward	Receive
PDU Composition	M	-	-
PDU Decomposition	M	-	M
Header Format Analysis	-	M	M
PDU Lifetime Control	-	M	I
Route PDU	-	M	-
Forward PDU	M	M	-
Segment PDU	M	(note 1)	-
Reassemble PDU	-	I	M
Discard PDU	-	M	M
Error Reporting	-	M	M
PDU Header Error Detection	M	M	M
Padding	(note 2)	(note 2)	(note 2)
Security	-	(note 3)	(note 3)
Complete Source Routing	-	(note 3)	-
Partial Source Routing	-	(note 4)	-
Record Route	-	(note 4)	-
QoS Maintenance	-	(note 4)	-

Table 9-1. Categorization of Functions

KEY:
M : Mandatory Function; must be implemented
\- : Not applicable
I : Implementation option, as described in text

Notes:

1) The Segment PDU function is in general mandatory for an intermediate system. However, a system which is to be connected only to subnetworks all offering the same maximum SNSDU size (such as identical Local Area Networks) will not need to perform this function and therefore does not need to implement it.

   If this function is not implemented, this shall be stated as part of the specification of the implementation.

2) The correct treatment of the padding function requires no processing. A conforming implementation shall support the function, to the extent of ignoring this parameter wherever it may appear.

3) This function may or may not be supported. If an implementation does not support this function, and the function is selected by a PDU, then the PDU shall be discarded, and an ER PDU shall be generated and forwarded to the originating network-entity if the Error Report flag is set.

4) This function may or may not be supported. If an implementation does not support this function, and the function is selected by a PDU, then the function is not provided and the PDU is processed exactly as though the function was not selected. The PDU shall not be discarded.

ANNEXES

(These annexes are provided for information for implementors and are not an integral part of the body of the Standard.)

ANNEX A. SUPPORTING TECHNICAL MATERIAL

A.1 Data Unit Lifetime

There are two primary purposes of providing a PDU lifetime capability in the ISO 8473 Protocol. One purpose is to ensure against unlimited looping of protocol data units. Although the routing algorithm should ensure that it will be very rare for data to loop, the PDU lifetime field provides additional assurance that loops will be limited in extent.

The other important purpose of the lifetime capability is to provide for a means by which the originating network entity can limit the Maximum NSDU lifetime. ISO Transport Protocol Class 4 assumes that there is a particular Maximum NSDU Lifetime in order to protect against certain error states in the connection establishment and termination phases. If a TPDU does not arrive within this time, then there is no chance that it will ever arrive. It is necessary to make this assumption, even if the Network Layer does not guarantee any particular upper bound on NSDU lifetime. It is much easier for Transport Protocol Class 4 to deal with occasional lost TPDUs than to deal with occasional very late TPDUs. For this reason, it is preferable to discard very late TPDUs than to deliver them. Note that NSDU lifetime is not directly associated with the retransmission of lost TPDUs, but relates to the problem of distinguishing old (duplicate) TPDUs from new TPDUs.

Maximum NSDU Lifetime must be provided to transport protocol entity in units of time; a transport entity cannot count "hops". Thus NSDU lifetime must be calculated in units of time in order to be useful in determining Transport timer values.

In the absence of any guaranteed bound, it is common to simply guess some value which seems like a reasonable compromise. In essence one is simply assuming that "surely no TPDU would ever take more than 'x' seconds to traverse the network." This value is probably chosen by observation of past performance, and may

vary with source and destination.

Three possible ways to deal with the requirement for a limit on the maximum NSDU lifetime are: (1) specify lifetime in units of time, thereby requiring intermediate systems to decrement the lifetime field by a value which is an upper bound on the time spent since the previous intermediate system, and have the Network Layer discard protocol data units whose lifetime has expired; (2) provide a mechanism in the Transport Layer to recognize and discard old TPDUs; or (3) ignore the problem, anticipating that the resulting difficulties will be rare. Which solution should be followed depends in part upon how difficult it is to implement solutions (1) and (2), and how strong the transport requirement for a bounded time to live really is.

There is a problem with solution (2) above, in that transport entities are inherently transient. In case of a computer system outage or other error, or in the case where one of the two endpoints of a connection closes without waiting for a sufficient period of time (approximately twice Maximum NSDU Lifetime), it is possible for the Transport Layer to have no way to know whether a particular TPDU is old unless globally synchronized clocks are used (which is unlikely). On the other hand, it is expected that intermediate systems will be comparatively stable. In addition, even if intermediate systems do fail and resume processing without memory of the recent past, it will still be possible (in most instances) for the intermediate system to easily comply with lifetime in units of time, as discussed below.

It is not necessary for each intermediate system to subtract a precise measure of the time that has passed since an NPDU (containing the TPDU or a segment thereof) has left the previous intermediate system. It is sufficient to subtract an upper bound on the time taken. In most cases, an intermediate system may simply subtract a constant value which depends upon the typical near-maximum delays that are encountered in a specific subnetwork. It is only necessary to make an accurate estimate on a per NPDU basis for those subnetworks which have both a relatively large maximum delay, and a relatively large variation in delay.

As an example, assume that a particular local area network has short average delays, with overall delays generally in the 1 to 5

millisecond range and with occasional delays up to 20 milliseconds. In this case, although the relative range in delays might be large (a factor of 20), it would still not be necessary to measure the delay for actual NPDUs. A constant value of 20 milliseconds (or more) can be subtracted for all delays ranging from .5 seconds to .6 seconds (.5 seconds for the propagation delay, 0 to .1 seconds for queueing delay) then the constant value .6 seconds could be used.

If a third subnetwork had normal delays ranging from .1 to 1 second, but occasionally delivered an NPDU after a delay of 15 seconds, the intermediate system attached to this subnetwork might be required to determine how long it has actually take the PDU to transit the subnetwork. In this last example, it is likely to be more useful to have the intermediate systems determine when the delays are extreme ad discard very old NPDUs, as occasional large delays are precisely what causes the Transport Protocol the most trouble.

In addition to the time delay within each subnetwork, it is important to consider the time delay within intermediate systems. It should be relatively simple for those gateways which expect to hold on to some data-units for significant periods of time to decrement the lifetime appropriately.

Having observed that (i) the Transport Protocol requires Maximum NSDU to be calculated in units of time; (ii) in the great majority of cases, it is not difficult for intermediate systems to determine a valid upper bound on subnetwork transit time; and (iii) those few cases where the gateways must actually measure the time take by a NPDU are precisely the cases where such measurement truly needs to be made, it can be concluded that NSDU lifetime should in fact be measured in units of time, and that intermediate systems should required to decrement the lifetime field of the ISO 8473 Protocol by a value which represents an upper bound on the time actually taken since the lifetime field was last decremented.

A.2 Reassembly Lifetime Control

In order to ensure a bound on the lifetime of NSDUs, and to effectively manage reassembly buffers in the Network Layer, the Reassembly Function described in Section 6 must control the

lifetime of segments representing partially assembled PDUs. This annex discusses methods of bounding reassembly lifetime and suggests some implementation guidelines for the reassembly function.

When segments of a PDU arrive at a destination network-entity, they are buffered until an entire PDU is received, assembled, and passed to the PDU Decomposition Function. The connectionless Internetwork Protocol does not guarantee the delivery of PDUs; hence, it is possible for some segments of a PDU to be lost or delayed such that the entire PDU cannot be assembled in a reasonable length of time. In the case of loss of a PDU "segment", for example, this could be forever. There are a number of possible schemes to prevent this:

a) Per-PDU reassembly timers,

b) Extension of the PDU Lifetime control function, and

c) Coupling of the Transport Retransmission timers.

Each of these methods is discussed in the subsections which follow.

A.2.1 Method (a)

assigns a "reassembly lifetime" to each PDU received and identified by its Data-unit Identifier. This is a local, real time which is assigned by the reassembly function and decremented while some, but not all segments of the PDU are being buffered by the destination network-entity. If the timer expires, all segments of the PDU are discarded, thus freeing the reassembly buffers and preventing a "very old" PDU from being confused with a newer one bearing the same Data-unit Identifier. For this scheme to function properly, the timers must be assigned in such a fashion as to prevent the phenomenon of Reassembly Interference (discussed below). In particular, the following guidelines should be followed:

1) The Reassembly Lifetime must be much less than the maximum PDU lifetime of the network (to prevent the confusion of old and new data-units).

2) The lifetime should be less than the Transport protocol's retransmission timers minus the average transit time of the network. If this is not done, extra buffers are tied up holding data which has alrèady been retransmitted by the Transport Protocol. (Note that an assumption has been made that such timers are integral to the Transport Protocol, which in some sense, dictates that retransmission functions must exist in the Transport Protocol employed).

A.2.2 Method (b)

is feasible if the PDU lifetime control function operates based on real or virtual time rather than hop-count. In this scheme, the lifetime field of all PDU segments of a Data-unit continues to be decremented by the reassembly function of the destination network-entity as if the PdU were still in transit (in a sense, it still is). When the lifetime of any segment of a partially reassembled PDU expires, all segments of that PDU are discarded. This scheme is attractive since the delivery behavior of the ISO 8473 Protocol would be identical for segmented and unsegmented PDUs.

A.2.3 Method (c)

couples the reassembly lifetime directly to the Transport Protocol's retransmission timers, and requires that Transport Layer management make known to Network Layer Management (and hence, the Reassembly Function) the values of its retransmission timers for each source from which it expects to be receiving traffic. When a PDU segment is received from a source, the retransmission time minus the anticipated transit time becomes the reassembly lifetime of that PDU. If this timer expires before the entire PDU has been reassembled, all segments of the PDU are discarded. This scheme is attractive since it has a low probability of holding PDU segments that have already been retransmitted by the source Transport-entity; it has, however, the disadvantage of depending on reliable operation of the Transport Protocol to work effectively. If the retransmission timers are not set correctly, it is possible that all PDUs would be discarded too soon, and the Transport Protocol would make no progress.

A.3 The Power of the Header Error Detection Function

A.3.1 General

The form of the checksum used for PDU header error detection is such that it is easily calculated in software or firmware using only two additions per octet of header, yet it has an error detection power approaching (but not quite equalling) that of techniques (such as cyclic polynomial checks) which involve calculations that are much more time- or space-consuming. This annex discusses the power of this error detection function.

The checksum consists of two octets, either of which can assume any value except zero. That is, 255 distinct values for each octet are possible. The calculation of the two octets is such that the value of either is independent of the value of the other, so the checksum has a total of 255 x 255 = 65025 values. If one considers all ways in which the PDU header might be corrupted as equally likely, then there is only one chance in 65025 that the checksum will have the correct value for any particular corruption. This corresponds to 0.0015 of all possible errors.

The remainder of this annex considers particular classes of errors that are likely to be encountered. The hope is that the error detection function will be found to be more powerful, or at least no less powerful, against these classes as compared to errors in general.

A.3.2 Bit Alteration Errors

First considered are classes of errors in which bits are altered, but no bits are inserted nor deleted. This section does not consider the case where the checksum itself is erroneously set to be all zero; this case is discussed in section A.3.4.

A burst error of length b is a corruption of the header in which all of the altered bits (no more than b in number) are within a single span of consecutively transmitted bits that is b bits long. Checksums are usually expected to do well against burst errors of a length not exceeding the number of bits in the header error detection parameter (16 for the PDU header). The PDU header error detection parameter in fact fails to detect only 0.000019 of all such errors, each distinct burst error of length 16 or less being considered to be equally likely. In particular,

it cannot detect an 8-bit burst in which an octet of zero is altered to an octet of 255 (all bits = 1) or vice versa. Similarly, it fails to detect the swapping of two adjacent octets only if one is zero and the other is 255.

The PDU header error detection, as should be expected, detects all errors involving only a single altered bit.

Undetected errors involving only two altered bits should occur only if the two bits are widely separated (and even then only rarely). The PDU header error detection detects all double bit errors for which the spacing between the two altered bits is less than 2040 bits = 255 octets. Since this separation exceeds the maximum header length, all double bit errors are detected.

The power to detect double bit errors is an advantage of the checksum algorithm used for the protocol, versus a simple modulo 65536 summation of the header split into 16 bit fields. This simple summation would not catch all such double bit errors. In fact, double bit errors with a spacing as little as 16 bits apart could go undetected.

A.3.3 Bit Insertion/Deletion Errors

Although errors involving the insertion or deletion of bits are in general neither more nor less likely to go undetected than are all other kinds of general errors, at least one class of such errors is of special concern. If octets, all equal to either zero or 255, are inserted at a point such that the simple sum C0 in the running calculation (described in Annex C) happens to equal zero, then the error will go undetected. This is of concern primarily because there are two points in the calculation for which this value for the sum is not a rare happenstance, but is expected; namely, at the beginning and the end. That is, if the header is preceded or followed by inserted octets all equal to zero or 255 then no error is detected. Both cases are examined separately.

Insertion of erroneous octets at the beginning of the header completely misaligns the header fields, causing them to be misinterpreted. In particular, the first inserted octet is interpreted as the network layer protocol identifier, probably eliminating any knowledge that the data unit is related to the

ISO 8473 Protocol, and thereby eliminating any attempt to perform the checksum calculation or invoking a different form of checksum calculation. An initial octet of zero is reserved for the Inactive Network Layer Protocol. This is indeed a problem but not one which can be ascribed to the form of checksum being used. Therefore, it is not discussed further here.

Insertion of erroneous octets at the end of the header, in the absence of other errors, is impossible because the length field unequivocally defines where the header ends. Insertion or deletion of octets at the end of the header requires an alteration in the value of the octet defining the header length. Such an alteration implies that the value of the calculated sum at the end of the header would not be expected to have the dangerous value of zero and consequently that the error is just as likely to be detected as is any error in general.

Insertion of an erroneous octet in the middle of the header is primarily of concern if the inserted octet has either the value zero or 255, and if the variable CO happens to have the value zero at this point. In most cases, this error will completely destroy the parsing of the header, which will cause the data unit to e discarded. In addition, in the absence of any other error, the last octet of the header will be thought to be data. This in turn will cause the header to end in the wrong place. In the case where the header otherwise can parse correctly, the last field will be found to be missing. Even in the case where necessary, the length field is the padding option, and therefore not necessary, the length field for the padding function will be inconsistent with the header length field, and therefore the error can be detected.

A.3.4 Checksum Non-calculation Errors

Use of the header error detection function is optional. The choice of not using it is indicated by a checksum parameter value of zero. This creates the possibility that the two octets of the checksum parameter (neither of which is generated as being zero) could both be altered to zero. This would in effect be an error not detected by the checksum since the check would not be made. One of three possibilities exists:

1) A burst error of length sixteen (16) which sets the entire

checksum to zero. Such an error could not be detected; however, it requires a particular positioning of the burst within the header. [A calculation of its effect on overall detectability of burst errors depends upon the length of the header.]

2) All single bit errors are detected. Since both octets of the checksum field must be non-zero when the checksum is being used, no single bit error can set the checksum to zero.

3) Where each of the two octets of the checksum parameter has a value that is a power of two, such that only one bit in each equals one (1), then a zeroing of the checksum parameter could result in an undetected double bit error. Furthermore, the two altered bits have a separation of less than sixteen (16), and could be consecutive. This is clearly a decline from the complete detectability previously described.

Where a particular administration is highly concerned about the possibility of accidental zeroing of the checksum among data units within its domain, then the administration may impose the restriction that all data units whose source or destination lie within its domain must make use of the header error detection function. Any data units which do not could be discarded, nor would they be allowed outside the domain. This protects against errors that occur within the domain, and would protect all data units whose source or destination lies within the domain, even where the data path between all such pairs crosses other domains (errors outside the protected domain notwithstanding).

ANNEX B. NETWORK MANAGEMENT

The following topics are considered to be major components of Network Layer management:

A. Routing

Considered by many to be the most crucial element of Network Layer management, since management of the Routing algorithms for networking seem to be an absolutely necessary prerequisite to a practical networking scheme.

Routing management consists of three parts; forwarding, decision, and update. Management of forwarding is the process of interpreting the Network Layer address to properly forward NSDUs on its next network hop on a route through the network. Management of decision is the process of choosing routes for either connections or NSDUs, depending on whether the network is operating a connection-oriented or connectionless protocol. The decision component will be driven by a number of considerations, not the least of which are those associated with Quality of Service. Management of update is the management protocol(s) used to exchange information among intermediate-systems/network- entities which is used in the decision component to determine routes.

To what extent is it desirable and/or practical to pursue a single OSI network routing algorithm and associated Management protocol(s)? It is generally understood that it is impractical to expect ISO to adopt a single global routing algorithm. On the other hand, it is recognized that having no standard at all upon which to make routing decisions effectively prevents an internetwork protocol from working at all. One possible compromise would be to define the principles for the behavior of an internetwork routing algorithm. A possible next step would be to specify the types of information that must be propagated among the intermediate-systems/network-entities via their update procedures. The details of the updating protocol might then be left to bilateral agreements among the cooperating administrations.

B. Statistical Analysis

These management functions relate to the gathering and reporting of information about the real-time behavior of the global network. They consist of Data counts such as number of PDUs forwarded, entering traffic, etc., and Event Counts such as topology changes, quality of service changes, etc.

C. Network Control

These management functions are those related to the control of the global network, and possibly could be performed by a Network Control Center(s). The control functions needed are not al all clear. Neither are the issues relating to what organization(s) is/are responsible for the management of the environment. Should there be a Network Control Center distinct from those provided by the subnetwork administrations? What subnetwork management information is needed by the network management components to perform their functions?

D. Directory Mapping Functions

Does the Network layer contain a Directory function as defined in the Reference Model? Current opinion is that the Network Layer restricts itself to the function of mapping NSAP addresses to routes.

E. Congestion Control

Does this come under the umbrella of Network Layer management? How?

F. Configuration Control

This is tightly associated with the concepts of Resource Management, and is generally considered to be somehow concerned with the control of the resources used in the management of the global network. The resources which have to be managed are Bandwidth (use of subnetwork resources), Processor (CPU), and Memory (buffers). Where is the responsibility for resources assigned, and are they appropriate for standardization? It appears that these

functions are tightly related to how one signals changes in Quality of Service.

G. Accounting

What entities, administrations, etc., are responsible for network accounting? How does this happen? What accounting information, if any, is required from the subnetworks in order to charge for network resources? Who is charged? To what degree is this to be standardized?

ANNEX C. ALGORITHMS FOR PDU HEADER ERROR DETECTION FUNCTION

This Annex describes algorithm which may be used to computer, check and update the checksum field of the PDU Header in order to provide the PDU Header Error Detection function described in Section 6.11.

C.1 Symbols used in algorithms

C0,C1	variables used in the algorithms
i	number (i.e., position) of an octet within the header
n	number (i.e., position) of the first octet of the checksum parameter (n=8)
L	length of the PDU header in octets
X	value of octet one of the checksum parameter
Y	value of octet two of the checksum parameter
a	octet occupying position i of the PDU header

C.2 Arithmetic Conventions

Addition is performed in one of the two following modes:

a) modulo 255 arithmetic;

b) eight-bit one's complement arithmetic in which, if any of the variables has the value minus zero (i.e., 255) it shall be regarded as though it was plus zero (i.e., 0).

C.3 Algorithm for Generating Checksum Parameters

A: Construct the complete PDU header with the value of the checksum parameter field set to zero;

B: Initialize C0 and C1 to zero;

C: Process each octet of the PDU header sequentially from i = 1 to L by

a) adding the value of the octet to C0; then

b) adding the value of C0 to C1;

D: Calculate X = (L-8)C0 - C1 (modulo 255) and Y = (L-7) (-C0) + C1 (modulo 255)

E: If X = 0, set X = 255;

F: If Y = 0, set Y = 255;

G: Place the values X and Y in octets 8 and 9 respectively.

C.4 Algorithm for Checking Checksum Parameters

A: If octets 8 and 9 of PDU header both contain 0 (all bits off), then the checksum calculation has succeeded; otherwise initialize C1 = 0, C0 - 0 and proceed;

B: process each octet of the PDU header sequentially from i = 1 to L by

a) adding the value of the octet to C0; then

b) adding the value of C0 to C1;

C: If, when all the octets have been processed, C0 = C1 = 0 (modulo 255) then the checksum calculation has succeeded; otherwise, the checksum calculation has failed.

C.5 Algorithm to adjust checksum parameter when an octet is altered

This algorithm adjusts the checksum when an octet (such as the lifetime field) is altered. Suppose the value in octet k is changed by Z = new_value - old_value.

If X and Y denote the checksum values held in octets n and n+1, respectively, then adjust X and Y as follows:

```
If X = 0 and Y = 0 do nothing, else;
     X := (k-n-1)Z + X (modulo 255) and
     Y := (n-k)Z + Y   (modulo 255).
If X is equal to zero, then set it to 255; and
similarly for Y.
```

For this Protocol, n = 8. If the octet being altered is the lifetime field, k = 4. For the case where the lifetime is decreased by 1 unit (Z = -1), the results simplify to

X := X + 5 (modulo 255) and
Y := Y - 4 (modulo 255).

Note:

To derive this result, assume that when octet k has the value Z added to it then X and Y have values ZX and ZY added to them. For the checksum parameters to satisfy the conditions of Section 6.11 both before and after the values are added, the following is required:

Z + ZX + ZY = 0 (modulo 255) and
(L-k+1)Z + (L-n+1)ZX + (L-n)ZY = 0 (modulo 255).

Solving these equations simultaneously yields ZX = (k-n-1)Z and ZY + (m-k)Z.